

MACHINE LEARNING AND ITS APPLICATION: A QUICK GUIDE FOR BEGINNERS



Indranath Chatterjee

Bentham Books

Machine Learning and Its Application: A Quick Guide for Beginners

Authored by

Indranath Chatterjee, Ph. D.

*Department of Computer Engineering
Tongmyong University
Busan
South Korea*

Machine Learning and Its Application: A Quick Guide for Beginners

Author: Indranath Chatterjee

ISBN (Online): 978-1-68108-940-9

ISBN (Print): 978-1-68108-941-6

ISBN (Paperback): 978-1-68108-942-3

© 2021, Bentham Books imprint.

Published by Bentham Science Publishers - Sharjah, UAE. All Rights Reserved.

Htuv'r wdrkj gf 'kp"42430

BENTHAM SCIENCE PUBLISHERS LTD.

End User License Agreement (for non-institutional, personal use)

This is an agreement between you and Bentham Science Publishers Ltd. Please read this License Agreement carefully before using the ebook/echapter/ejournal ("Work"). Your use of the Work constitutes your agreement to the terms and conditions set forth in this License Agreement. If you do not agree to these terms and conditions then you should not use the Work.

Bentham Science Publishers agrees to grant you a non-exclusive, non-transferable limited license to use the Work subject to and in accordance with the following terms and conditions. This License Agreement is for non-library, personal use only. For a library / institutional / multi user license in respect of the Work, please contact: permission@benthamscience.net.

Usage Rules:

1. All rights reserved: The Work is 1. the subject of copyright and Bentham Science Publishers either owns the Work (and the copyright in it) or is licensed to distribute the Work. You shall not copy, reproduce, modify, remove, delete, augment, add to, publish, transmit, sell, resell, create derivative works from, or in any way exploit the Work or make the Work available for others to do any of the same, in any form or by any means, in whole or in part, in each case without the prior written permission of Bentham Science Publishers, unless stated otherwise in this License Agreement.
2. You may download a copy of the Work on one occasion to one personal computer (including tablet, laptop, desktop, or other such devices). You may make one back-up copy of the Work to avoid losing it.
3. The unauthorised use or distribution of copyrighted or other proprietary content is illegal and could subject you to liability for substantial money damages. You will be liable for any damage resulting from your misuse of the Work or any violation of this License Agreement, including any infringement by you of copyrights or proprietary rights.

Disclaimer:

Bentham Science Publishers does not guarantee that the information in the Work is error-free, or warrant that it will meet your requirements or that access to the Work will be uninterrupted or error-free. The Work is provided "as is" without warranty of any kind, either express or implied or statutory, including, without limitation, implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the results and performance of the Work is assumed by you. No responsibility is assumed by Bentham Science Publishers, its staff, editors and/or authors for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products instruction, advertisements or ideas contained in the Work.

Limitation of Liability:

In no event will Bentham Science Publishers, its staff, editors and/or authors, be liable for any damages, including, without limitation, special, incidental and/or consequential damages and/or damages for lost data and/or profits arising out of (whether directly or indirectly) the use or inability to use the Work. The entire liability of Bentham Science Publishers shall be limited to the amount actually paid by you for the Work.

General:

1. Any dispute or claim arising out of or in connection with this License Agreement or the Work (including non-contractual disputes or claims) will be governed by and construed in accordance with the laws of the U.A.E. as applied in the Emirate of Dubai. Each party agrees that the courts of the Emirate of Dubai shall have exclusive jurisdiction to settle any dispute or claim arising out of or in connection with this License Agreement or the Work (including non-contractual disputes or claims).
2. Your rights under this License Agreement will automatically terminate without notice and without the

need for a court order if at any point you breach any terms of this License Agreement. In no event will any delay or failure by Bentham Science Publishers in enforcing your compliance with this License Agreement constitute a waiver of any of its rights.

3. You acknowledge that you have read this License Agreement, and agree to be bound by its terms and conditions. To the extent that any other terms and conditions presented on any website of Bentham Science Publishers conflict with, or are inconsistent with, the terms and conditions set out in this License Agreement, you acknowledge that the terms and conditions set out in this License Agreement shall prevail.

Bentham Science Publishers Ltd.

Executive Suite Y - 2
PO Box 7917, Saif Zone
Sharjah, U.A.E.
Email: subscriptions@benthamscience.net



CONTENTS

PREFACE	i
ABOUT THE AUTHOR	iv
DEDICATION	v
CHAPTER 1 INTRODUCTION	1
1.1. WHAT IS ARTIFICIAL INTELLIGENCE?	1
1.1.1. Evolution of AI.....	2
1.1.2. Dimensions of AI (Types of AI)	6
1.1.3. Why Is Learning AI important?	8
1.2. NEED FOR MACHINE LEARNING.....	9
1.3. WHAT IS LEARNING IN MACHINE LEARNING?	10
1.4. WHEN DO WE NEED MACHINE LEARNING?	11
1.5. TYPES OF LEARNING	13
1.5.1. Supervised	14
1.5.2. Unsupervised	14
1.5.3. Semi-supervised	15
1.5.4. Reinforcement	15
1.5.5. Self-supervised	15
1.6. WHAT IS THE NEED FOR THIS BOOK ON MACHINE LEARNING?	15
1.7. OUTLINE OF THE BOOK	16
CONCLUDING REMARKS	18
CHAPTER 2 SUPERVISED MACHINE LEARNING: CLASSIFICATION.....	19
2.1. INTRODUCTION TO SUPERVISED MACHINE LEARNING.....	19
2.1.1. Supervised Machine Learning	19
2.1.2. What is Classification?	20
2.1.3. Types of Classification	22
2.2. DECISION TREE	23
2.2.1. Overview	23
2.2.2. Algorithmic Framework	24
2.2.2.1. <i>Different Terminologies used in Decision Tree</i>	24
2.2.2.2. <i>Entropy</i>	25
2.2.2.3. <i>Information Gain</i>	25
2.2.2.4. <i>Gini Index</i>	26
2.2.3. Hands-on Example.....	27
2.2.3.1. <i>Building a Decision Tree with the Help of Information Gain</i>	27
2.2.4. Types of Decision Tree Algorithms	32
2.2.5. Advantages and Disadvantages of the Decision Tree	32
2.2.6. Programming Approach for Decision Tree.....	33
2.3. RANDOM FOREST	35
2.3.1. Overview	35
2.3.2. Why Use Random Forest?	36
2.3.3. Algorithmic Framework.....	37
2.3.3.1. <i>Assumptions for Random Forest</i>	37
2.3.4. Advantages and Disadvantages of Random Forest	38
2.3.4.1. <i>Advantages</i>	38
2.3.4.2. <i>Disadvantages</i>	38
2.3.5. Programming Approach for Random Forest.....	38
2.4. K-PEAREST NEIGHBOR	40
2.4.1. Overview	40
2.4.2. When Do We Use KNN Algorithm?	41
2.4.3. How to Select the Value of K?	45
2.4.4. Algorithmic Framework.....	45
2.4.4.1 <i>How Does KNN Work?</i>	45
2.4.4.2 <i>Pseudo Code of KNN</i>	46
2.4.5. Programming Approach For K-nearest Neighbor.....	46

2.5. NAÏVE BAYES CLASSIFIER	48
2.5.1. Overview	48
2.5.1.1. <i>Conditional Probability Model of Classification</i>	49
2.5.1.2. <i>Calculating the Prior and Conditional Probabilities</i>	50
2.5.2. Hands-on Example.....	51
2.5.2.1. <i>Make Predictions with Naïve Bayes</i>	51
2.5.3. Advantages and Disadvantages of Naïve Bayes:	54
2.5.3.1. <i>Advantages</i>	54
2.5.3.2. <i>Disadvantages</i>	55
2.5.4. Tips for Using Naive Bayes Algorithm	55
2.5.5. Programming Approach for Naïve Bayes Classifier.....	56
2.6. SUPPORT VECTOR MACHINE	58
2.6.1. Overview	58
2.6.1.1. <i>Decision Rule</i>	61
2.6.2. Hands-on Example.....	62
2.6.2.1. <i>Working of SVM</i>	62
2.6.3. The Kernel Trick.....	66
2.6.3.1. <i>Choosing a Kernel Function</i>	67
2.6.4. Advantages and Disadvantages of Support Vector Machines.....	68
2.6.5. Programming Approach for Support Vector Machine	69
CONCLUDING REMARKS	71
CHAPTER 3 UNSUPERVISED MACHINE LEARNING: CLUSTERING	72
3.1. INTRODUCTION TO UNSUPERVISED MACHINE LEARNING	72
3.1.1. What is Clustering?	72
3.1.2. Types of Clustering Methods.....	73
3.1.3. Real-life Applications of Clustering	73
3.2. K-MEANS CLUSTERING	74
3.2.1. Overview	74
3.2.2. Algorithmic Framework.....	74
3.2.2.1. <i>Introduction to K-means Algorithm</i>	74
3.2.3. Hands-on Example.....	76
3.2.4. Weakness of K-means Clustering	86
3.2.5. Strength and Application of K-means Clustering	86
3.2.6. Programming Approach for K-means Clustering	87
3.3. HIERARCHICAL CLUSTERING	90
3.3.1. Overview	90
3.3.2. Algorithmic Framework.....	91
3.3.3. Hands-on Example.....	94
3.3.4. Programming Approach for Hierarchical Clustering	100
3.4. SELF-ORGANIZING MAP	104
3.4.1. Overview	104
3.2.2.1. <i>How SOM Works?</i>	105
3.4.2. Algorithmic Framework.....	106
3.4.3. Advantage and Disadvantage of SOM	107
3.4.3.1. <i>Advantage</i>	107
3.4.3.2. <i>Disadvantage</i>	105
3.4.3.3. <i>Different Perspective of SOM</i>	108
3.4.4. Programming Approach for K-nearest Neighbor	108
CONCLUDING REMARKS	113
CHAPTER 4 REGRESSION: PREDICTION	114
4.1. INTRODUCTION TO REGRESSION.....	114
4.1.1. What is Regression?	114
4.1.1.1. <i>Linear Regression</i>	115
4.1.1.2. <i>Advantage</i>	115
4.1.2. How is it Different from Classification?	116
4.1.3. Applications of Regression	116
4.2. LINEAR REGRESSION	117
4.2.1. Overview.....	117

4.2.1.1. Simple Regression	118
4.2.1.2. Making a Prediction	119
4.2.1.3. Multi-variable Regression	120
4.2.2. Linear Regression Line	122
4.2.2.1. Positive Linear Relationship	122
4.2.2.2. Negative Linear Relationship	122
4.2.2.3. Assumptions in Regression and its Justification	123
4.2.3. Regression Algorithms	124
4.2.4. Programming Approach for Linear Regression	125
4.3. LOGISTIC REGRESSION	127
4.3.1. Overview	127
4.3.1.1. Comparison to Linear Regression	128
4.3.1.2. Binary Logistic Regression	129
4.3.1.3. Multiclass Logistic Regression	129
4.3.2. Algorithmic Framework	131
4.3.2.1. Predict with Logistic Regression	133
4.3.2.2. Data Preparation for Logistic Regression	134
4.3.3. Programming Approach for Logistic Regression	134
CONCLUDING REMARKS	137
 CHAPTER 5 REINFORCEMENT LEARNING	138
5.1. INTRODUCTION TO REINFORCEMENT LEARNING	138
5.1.1. Overview	139
5.1.1.1. Data Preparation for Logistic Regression	140
5.1.1.2. How RL Differs from Supervised Learning	140
5.1.2. Element of Reinforcement Learning	140
5.2. ALGORITHMIC FRAMEWORK	142
5.2.1. Basic Steps of Reinforcement Learning	142
5.2.2. Types of Reinforcement Learning	142
5.2.3. Elements of Reinforcement Learning	143
5.2.4. Models of Reinforcement Learning	145
5.2.4.1. Markov's Decision Process (MDP)	145
5.2.4.2. Q-learning	145
5.3. HANDS-ON EXAMPLE OF REINFORCEMENT LEARNING	156
5.4. REAL-WORLD EXAMPLES OF A REINFORCEMENT LEARNING TASK	158
5.4.1. Advantages	159
5.4.1. Disadvantages	160
5.5. PROGRAMMING APPROACH FOR REINFORCEMENT LEARNING	160
CONCLUDING REMARKS	169
 CHAPTER 6 DEEP LEARNING:A NEW APPROACH TO MACHINE LEARNING	170
6.1. INTRODUCTION TO REINFORCEMENT LEARNING	170
6.1.1. Architecture of Deep Learning	171
6.1.2. Working Principle of Deep Learning	172
6.1.2.1. Training Phase	172
6.1.3. Types of Deep Learning	174
6.1.4. Advantage and Disadvantage of Deep Learning	178
6.1.4.1. Advantages	178
6.1.4.2. Disadvantages	178
6.1.5. Application of Deep Learning	179
6.2. ARTIFICIAL NEURAL NETWORK	179
6.2.1. Overview	179
6.2.2. Perceptron Learning	183
6.2.2.1. Linear Threshold Unit (TLU)	185
6.2.2.2. Single Layer Perceptron Learning	186
6.2.2.3. Multilayer Perceptron Learning	188
6.2.3. Working Principle of Artificial Neural Network	190
6.2.4. Types of Neural Network	190
6.2.5. Applications of Neural Network	192
6.2.6. Programming Approach for Artificial Neural Network (ANN)	193

6.3. COMPONENTS OF NEURAL NETWORK	196
6.3.1. Layers	196
<i>6.3.1.1. Input Layer.....</i>	197
<i>6.3.1.2. Output Layer.....</i>	198
<i>6.3.1.3. Hidden Layer</i>	199
6.3.2. Weights and Bias of a Neuron	200
<i>6.3.2.1. Weights</i>	200
<i>6.3.2.2. Bias</i>	202
6.3.3. Activation Functions.....	202
<i>6.3.3.1. Multi-state Activation Functions</i>	203
<i>6.3.3.2. Identity Activation Functions</i>	204
<i>6.3.3.3. Binary Step Activation Functions.....</i>	205
<i>6.3.3.4. Sigmoid Activation Function.....</i>	206
<i>6.3.3.5. Tanh Activation Function.....</i>	206
<i>6.3.3.6. Rectified Linear Unit (Relu) Activation Function</i>	207
<i>6.3.3.7. Softmax Activation Function</i>	208
<i>6.3.3.8. Softplus Activation Function</i>	209
<i>6.3.3.9. Exponential Linear Unit Activation Function</i>	211
<i>6.3.3.10. Exponential Linear Unit Activation Function</i>	212
<i>6.3.3.11. Exponential Linear Unit Activation Function</i>	213
6.3.4. Forward Propagation	214
6.3.5. Backpropagation	215
6.3.6. Learning Rate	215
6.3.7. Gradient Descent	216
<i>6.3.7.1. Gradient</i>	217
<i>6.3.7.2. The General Idea</i>	218
<i>6.3.7.3. Type of Gradient Descent</i>	219
6.4. CONVOLUTIONAL NEURAL NETWORK.....	220
6.4.1. Overview	221
6.4.2. Algorithmic Framework.....	221
<i>6.4.2.1. Layers</i>	222
6.4.3. Types of CNN	228
6.4.4. Programming Approach for Convolutional Neural Network	229
6.5. RECURRENT NEURAL NETWORK	234
6.5.1. Overview	234
6.5.2. Algorithmic Framework.....	235
<i>6.5.2.1. Backpropagation through Time</i>	237
<i>6.5.2.2. Need for More than RNN: Vanishing and Exploding Gradient Problem</i>	239
<i>6.5.2.3. Types of RNN</i>	239
6.5.3. Hopfield Network	242
6.5.4. Long Short-term Memory (LSTM)	246
<i>6.5.4.1. Few Concepts in LSTM</i>	247
6.5.5. LSTM Hyper-parameter Tuning	249
6.5.6. Programming Approach for Recurrent Neural Network	250
CONCLUDING REMARKS	255
CHAPTER 7 FEATURE ENGINEERING	256
7.1. INTRODUCTION	256
7.1.1. Types of Feature Selection	257
7.2. FILTER-BASED APPROACH: HYPOTHESIS TESTING	259
7.2.1. T-test.....	260
<i>7.2.1.1. Hypothesis.....</i>	262
<i>7.2.1.2. Tables of T-distribution</i>	264
<i>7.2.1.3. T-score</i>	264
<i>7.2.1.4. P-value</i>	265
<i>7.2.1.5. Degree of Freedom</i>	265
7.2.2. Z-test.....	265
<i>7.2.2.1. Z-score Mean</i>	265
<i>7.2.2.2. What Exactly is the Central Limit Theorem?</i>	266

7.2.2.3. When to Perform a Z Test?	266
7.2.2.4. Difference Between T-test and Z-test	267
7.2.3. ANOVA.....	268
7.2.4. MANOVA	269
7.2.4.1. Assumptions	270
7.2.4.2. Exceptional Situations	270
7.2.4.3. MANOVA vs. ANOVA	271
7.3. FILTER-BASED APPROACH: CORRELATION.....	271
7.3.1. Correlation Analysis	272
7.3.2. Pearson's Correlation	272
7.3.2.1. Correlation Coefficient	273
7.3.2.2. Assumptions	273
7.3.2.3. The Cramer's V Correlation	273
7.3.3. Chi-square Test.....	274
7.3.3.1. Chi-Square P-values	275
7.3.3.2. Algorithm for Chi-square Test	275
7.3.3.3. Use of Chi-square Test	276
7.3.4. Spearman's Rank Correlation.....	276
7.4. EVOLUTIONARY ALGORITHMS.....	278
7.4.1. Genetic Algorithm	278
7.4.1.1. Use of Chi-square Test	279
7.4.1.2. Search Space	279
7.4.1.3. Genetic Operators	279
7.4.1.4. Multi-objective Functions	282
7.4.1.5. Termination	282
7.4.1.6. Algorithmic Framework	282
7.4.2. Particle Swarm Optimization.....	282
7.4.2.1. Particles	284
7.4.2.2. Swarms	284
7.4.2.3. Optimization	285
7.4.3. ANT Colony Optimization	286
7.4.3.1. Algorithmic Framework	288
7.4.3.2. Application of Ant Colony Optimization	288
CONCLUDING REMARKS	289
CHAPTER 8 APPLICATIONS OF MACHINE LEARNING AND DEEP LEARNING.....	290
8.1. INTRODUCTION	290
8.2. PATTERN RECOGNITION	291
8.2.1. Face Recognition	291
8.2.1.1. Python Implementation	292
8.2.2. Optical Character Recognition.....	298
8.2.2.1. Python Implementation	299
8.2.3. Object Recognition	301
8.2.3.1. Python Implementation	302
8.3. VIDEO PROCESSING.....	308
8.3.1. Video Processing for Object Detection.....	308
8.3.1.1. Python Implementation	309
8.4. MEDICAL IMAGING	312
8.4.1. Neuroimaging Application With fMRI.....	313
8.4.1.1. Python Implementation	314
8.4.2. Tumor Detection Using MRI	318
8.4.2.1. Python Implementation	319
8.5. COMPUTATIONAL LINGUISTIC.....	324
8.5.1. Sentiment Analysis Using NLP	325
8.5.1.1. Python Implementation	326
8.6. LIST OF POSSIBLE REAL-WORLD APPLICATIONS	330
8.6.1. Image Recognition	330
8.6.2. Sentiment Analysis	330
8.6.3. Speech Recognition	330

8.6.4. Medical Diagnosis	330
8.6.5. News Classification	330
8.6.6. Predictive Analysis	330
8.6.7. Video Surveillance	331
8.6.8. Text Extraction	331
8.6.9. Email Categorization	331
8.6.10. Social Media Analysis	331
CONCLUDING REMARKS	331
CHAPTER 9 CONCLUSIONS	332
CHAPTER 10 REFERENCES	335
SUBJECT INDEX	337

PREFACE

Over the past two decades, the evolution of Machine Learning has risen to a great extent. With the invention of Artificial Intelligence, things were getting more comfortable and accessible. Artificial intelligence needs a system to be fed with pre-defined conditional statements to perform some tasks on behalf of human beings. Gradually human needs a more stable and autonomous system that can learn on its own. There comes a lack of another technology that drastically changes the concept of artificial intelligence. With the invention of machine learning, advancements are going on every single day. With ever-increasing data sources and automated computation, technologies based on machine learning are coming alive very often.

The purpose of writing 'another book on machine learning is always a challenging task to attract the reader's attention. Machine learning is the most discussed topic of this decade and for a few more decades. The basic knowledge of machine learning is very needed. Most people are unaware of the fundamental theories and applications of machine learning.

Among many books available in the market on this topic, this book targets reaching all the corners of the reading society. From naïve learners to professional machine learning experts will find this book handy and helpful for everyday application. Most books are written in challenging mathematical perspectives, which pose incomprehensibility for most readers, especially students and industry engineers.

This book aims to cover most of the Machine Learning curriculum prescribed in most of the top universities. It also covers advanced topics like Deep Learning and Feature Engineering. This book's added feature is the entire chapter on real-world machine learning applications using Python programming, which will be truly beneficial for all the researchers and engineers, with open-ended ideas on new problems and their solutions in a Pythonic way.

This book is written effortlessly and straightforwardly with enriched theories and more minor mathematical complications, but more easily comprehensive application aspects. In every chapter, topics are described in such a way, keeping in mind readers from all sections. Every topic and subtopic is described with examples and Python code snippets for a more accessible explanation. The chapters are presented with a well-explained illustration and flowchart for a better

understanding of the topic. Thus, this book on machine learning will surely catch the beginners' attention in the Machine Learning domain. The audience will include, University students, Young Researchers, Ph.D. students, Professors, and software engineers who want to gain knowledge in Machine learning from scratch. I believe this book will be in demand of most of the University libraries and bookstores.

CONSENT FOR PUBLICATION

Not applicable.

CONFLICT OF INTEREST

A single author entirely writes this book. So, the conflict of interest does not apply to this book.

ACKNOWLEDGEMENTS

Writing a book is more exciting than I anticipated and more rewarding than I could have dreamed. Nothing would have been imaginable without the strength and ability bestowed upon me by the Almighty God because I would not be able to do anything without Him. I want to thank and express my gratitude to my closest family and friends.

I am eternally grateful to my family for their unwavering mental support and persistent encouragement, which has aided me in accomplishing this book in a timely and efficient manner. My family members, Mr. Narendranath Chatterjee, Mrs. Rupasree Chatterjee, Mr. Rudranath Chatterjee, and Ms. Soumi Chatterjee, deserve special thanks for sticking by my side throughout the duration. Love to you all!

I want to express intense gratitude to my dearest people, Mr. Ajay Kumar and Mrs. Rekha Devi, for their unceasing impetus and motivation. A huge cheer to you!

I am eternally grateful to my professor, friend, and charioteer, Prof. Naveen Kumar, whose insightful guidance and wisdom drove me to refine my belief to evolve the best in me.

A special thanks to my students, without whose contribution, this book may not look so magnificent as it is now. I am grateful to my dearest student Ms. Videsha Bansal for her continuous support and contribution in organizing the contents. I am very thankful to my beloved student at Tongmyong University, Mr. Sunghyun Kim, for his contribution and support in the programming section of the book. I am also thankful to my beloved research student Ms. Lea Baumgärtner for her encouragement and assistance in the programming section of the book. A special thanks to my student Mr. Sajal Jain for supporting me in giving the professional touch to the figures and diagrams.

I am very much grateful to Prof. Pamela Douglas from University of California, Los Angeles for reviewing the book and giving her valuable comments to improve the overall content of this book. I am also thankful to all my friends and colleagues at Tongmyong University for their support and good wishes. I am grateful to Mrs. Humaira Hashmi, Editorial Manager of Bentham Science Publishers, for her continuous support during the period.

Indranath Chatterjee, Ph. D.
Department of Computer Engineering
Tongmyong University
Busan
South Korea

ABOUT THE AUTHOR

Dr. Indranath Chatterjee is currently working as a Professor at the Department of Computer Engineering, Tongmyong University, Busan, South Korea. He has done his Ph. D. in Computational Neuroscience from the Department of Computer Science, University of Delhi, Delhi, India. In teaching, his areas of expertise are Artificial Intelligence, Machine Learning, Natural Language Processing, Deep Learning, and Data Science. In research, his keen interests lie in Computational Neuroscience, Medical Imaging, Schizophrenia, fMRI, Advanced Machine Learning, and Big Data Analytics. He is currently the author of eight textbooks on Computer Science and Neuroscience published numerous research papers in renowned international journals and conferences. Till date, he has been conferred with several awards from various international associations and conferences. He is currently serving as a Chief Section Editor and Editor of several international journals of high repute. Being an expert in Open-Science, he is also serving as a member of the Advisory board of various international Open-Science organizations worldwide. He is presently working on several projects of government & non-government organizations as PI/co-PI, related to neuroscience and machine learning for a broader societal impact, in collaboration with several universities globally. He is an active professional member of the Association of Computing Machinery (ACM, USA), Organization of Human Brain Mapping (OHBM, USA), Federations of European Neuroscience Society (FENS, Belgium), International Neuroinformatics Coordinating Facility (INCF, Sweden), and Association for Clinical Neurology and Mental Health (ACNM, India).

.

DEDICATION

**This book is dedicated to my parents,
Maa and Babai.
Thank you for everything... ☺**

There is no knowledge in nature; all knowledge comes from the human soul. Man manifests knowledge, discovers it within himself, which is pre-existing through eternity.

Swami Vivekananda

CHAPTER 1

Introduction to Machine Learning

Abstract: The introduction chapter summarizes various necessary topics on artificial intelligence and machine learning. It introduces the readers with a strong foundation on the basic concepts of advanced usage of artificial intelligence, which led to the beginning of an era of machine learning. Here, the readers will learn different aspects of machine learning concisely which are described in detail in the coming chapters. Firstly, this chapter discusses the dimensions of AI and the reasons for the transition of traditional AI to modern-day machine learning techniques. Secondly, it discusses the benefits of machine learning in day-to-day life. Thirdly, it discusses the machine learning algorithms' main pillars for training and developing prediction models. Finally, this chapter will give a broad outline of this book in a concise chapter-wise description.

Keywords: Artificial Intelligence (AI), Evolution of AI, Learning, Machine Learning, Self-supervised Learning, Supervised Learning, Unsupervised Learning.

1.1. WHAT IS ARTIFICIAL INTELLIGENCE?

Artificial intelligence or AI refers to the simulation of human intelligence of computers designed to think and imitate their behavior like humans. The word can also be extended to any system that shows features linked to a human mind, such as understanding and problem-solving.

When most people use the word artificial intelligence, robots are usually the first to come to mind. In this way, big-budget films and novels have created a story about a robot that destroys earth and humans. However, it is not far from the facts.

Authors Stuart Russell and Peter Norvig discuss the topic in their pioneering textbook *Artificial Intelligence: A Modern Approach* [1] by unifying their work around the theme of intelligent agents in computers. The authors described AI as:

"the study of agents that receive percepts from the environment and perform actions" (Russel and Norvig viii) [1].

Artificial intelligence originated from the idea that it is possible to describe human intelligence so that a computer can effectively imitate it and perform tasks, from the easiest to those that are much more complicated. Artificial intelligence's purposes include knowledge, logic, and understanding.

Previous standards that described artificial intelligence became obsolete as technology progresses. Machines that measure fundamental functions or identify text through optical character recognition, for example, are no longer thought to embody AI since this function is now taken for granted as an innate machine function.

To support several varied sectors, AI is constantly developing. AI is everywhere using a cross-disciplinary approach focused on math, computer science, linguistics, psychology, and more.

Artificial intelligence's perfect feature is the capacity to rationalize and perform decisions with the most excellent chance of fulfilling a particular purpose. Machine learning refers to the impression that computer programs can automatically learn from and respond to new data without human help. It is a branch of artificial intelligence. Deep learning techniques make this automated learning by ingesting vast quantities of unstructured data such as text, images, or video.

1.1.1. Evolution of AI

Artificial intelligence is certainly not another idea; its narrating roots go as far back as Greek relics. Notwithstanding, it was not precisely a century before the mechanical upset took off, and AI went from fiction to truly conceivable reality.

In the initial portion of the twentieth century, sci-fi acquainted the world with the idea of robots. By the 1950s, we had an age of researchers, mathematicians, and scholars who were socially acclimatized in their brains with the idea of human-made consciousness (or artificial intelligence). One such individual was Alan Turing, a British scientist who investigated the numerical chance of human-made consciousness. Turing recommended that people utilize accessible data to take care of issues and decide why the machines cannot do something very similar? This was the consistent system of his 1950 paper, "Computing Machinery and Intelligence," in which he examined how to assemble intelligent machines and test their knowledge.

From 1957 to 1974, AI got importance. Personal computers could store more data and turned out to be fast, less expensive, and more available. AI calculations additionally improved, and individuals improved at knowing which calculation to apply to their concerns. Early exhibits, for example, Newell and Simon's General Problem Solver and Joseph Weizenbaum's ELIZA, showed promising results toward the objectives of critical thinking and the translation of communication in language individually. These victories, just as the promotion of driving specialists (particularly the participants of the DSRPAI) persuaded government offices, for example, the Defense Advanced Research Projects Agency (DARPA), to subsidize AI research few organizations. The public authority was especially inspired by a machine that could interpret and decipher communication in language just as high throughput information handling. Idealism was high, and assumptions were considerably higher. In 1970 Marvin Minsky revealed to Life Magazine, "from three to eight years we will have a machine with the general intelligence of an average human being." However, while the essential rule verification was there, there was far to go before the ultimate objectives of standard language preparation, conceptual reasoning, and self-acknowledgment could be accomplished.

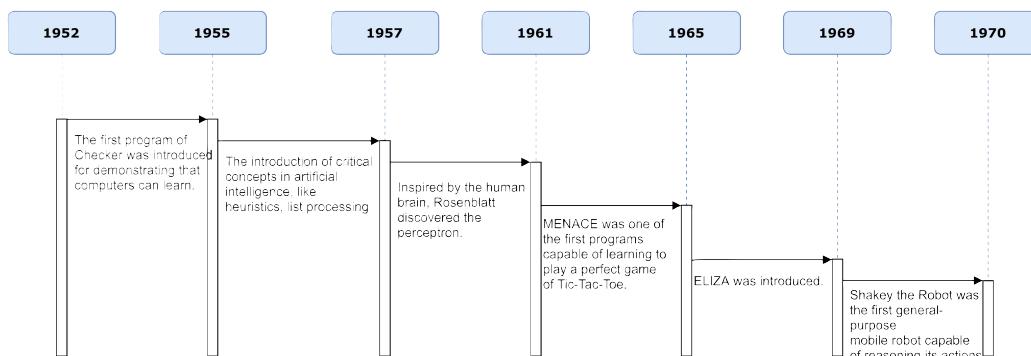


Fig. (1.1). Evolution of AI from 1950's to 1970's.

As we see in Fig. (1.1), it shows the evolution of artificial intelligence from the year 1952 to 1970.

The great scientist Simon and Minsky stated AI as:

Herbert Simon stated in 1957 that "*It is not my aim to surprise or shock you, but the simplest way I can summarize is to say that there are now in the world machines that think, that learn, and that create. Moreover, their ability to do these things is*

going to increase rapidly until – in a visible future – the range of problems they can handle will be coextensive with the range to which the mind has been applied”.

Marvin Minsky stated in 1970 that “*In three to eight years, we will get a machine with average human intelligence*”.

During the 1980s, the AI outlook changed to emblematic AI and the supposed “expert system” or “information-based frameworks.” The fundamental thought was to get human master information in a PC structure and spread it as a program to numerous computers.

During the 1990s, organizations began delivering expert systems frameworks. These organizations offered programming bundles called “inference engines” and related information administrations to the clients. As in Fig. (1.2), we see the evolution of artificial intelligence from 1968 to 1990.

Expert systems had two parts:

1. The information base – an assortment of realities, facts, rules, and connections on a particular field;
2. The inference tool that depicted how to control and consolidate these images.

Current realities and rules had unequivocal portrayals and were modifiable. Lisp and Prolog were the principal representative programming languages.

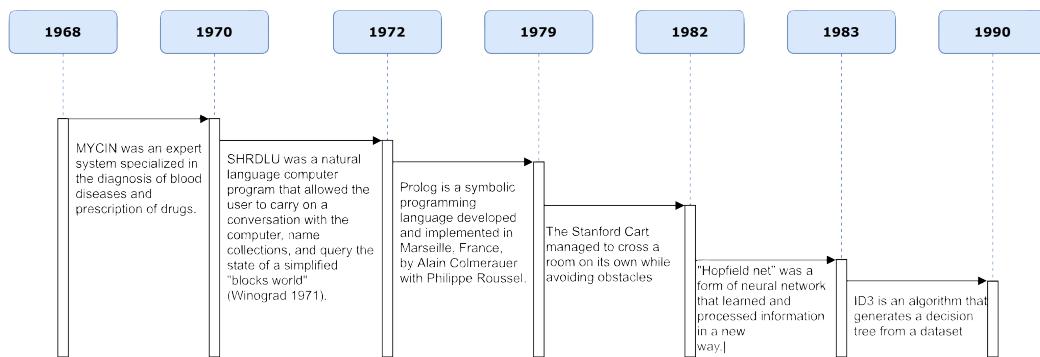


Fig. (1.2). Evolution of AI from the 1970s to 1990s.

During the 1990s–2010s, AI tended to complex problems, making arrangements discovered valuable in diverse application spaces, including information mining, modern advanced mechanics, coordinations, business knowledge, banking programming, clinical analysis, proposal frameworks, and web indexes. Artificial intelligence specialists started to create more advanced numerical methods. There was a far-reaching acknowledgment that numerous AI issues have effectively been dealt with by specialists in fields like mathematics, economics, and the area of operation research. The mathematics behind it permitted a more significant joint effort with set up fields and made AI a complete logical control.

Numerous AI analysts during the 1990s intentionally called their work by different names, like informatics, knowledge-based frameworks, cognitive frameworks, optimization algorithm, or computational intelligence. The new names assisted with obtaining financing.

In 2006, Professor Fei-Fei Li at Stanford University contributed a drastic change to AI thinking. The core insight of the scientist around then was that "If you can't even do one image well, why to try with thousands or tens of thousands of images?" Her theory was that AI's principle stated the information amount that affects the present reality and stated that more information or data would deliver better models. In 2009, ImageNet was introduced containing 3.2 million pictures, all labeled, isolated into 5,247 classes, organized into 12 subtypes like "well-evolved creature", "vehicle", "furniture", and so on. In Fig. (1.3), we see the evolution of artificial intelligence from 1993 till today (specifically 2019).

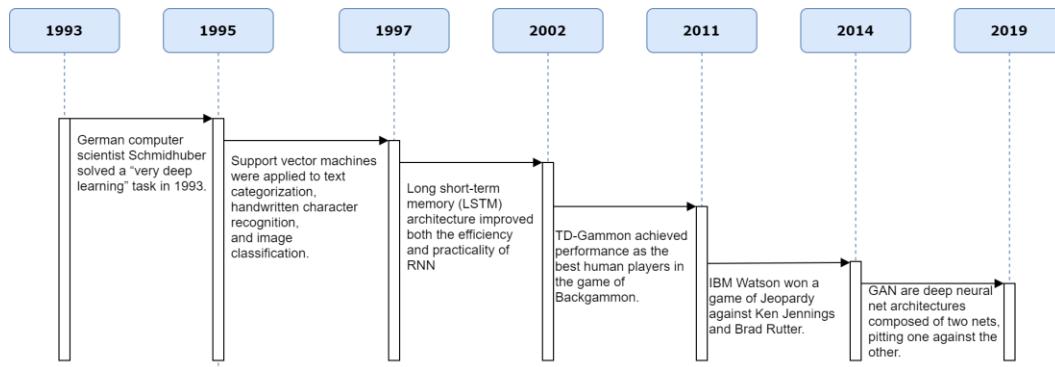


Fig. (1.3). Evolution of AI from 1990's to 2020's.

1.1.2. Dimensions of AI (Types of AI)

Exploration shows that 40% of organizations use artificial intelligence (AI) but lack unmistakable proof of its application. Without proper knowledge and understanding of this field, that develops increasingly, based on the scope of popular expressions in AI such as data mining, machine learning, and deep learning.

What's more, unavoidably, with disarray comes underperformance. Appropriate comprehension of AI and guaranteeing its precise execution is particularly significant among new companies. Companies that are named as being "in the field of AI" can draw in 15-50 percent more in their profit than other tech organizations, exhibiting the force of its importance [2].

When discussing the dimensions of AI, we mostly talk about six dimensions of AI, which are as follows:

- a) Automation: programmed dynamic response learning and disclosure by utilizing information.
- b) Augmentation: enlargement of the insight of existing items.
- c) Adaptation: adjusting by utilizing information.
- d) Analysis: finding other truth by dissecting information.
- e) Accuracy: expanding precision of existing or new framework.
- f) Acquisition procuring more out of available information.

We need to accomplish more than just instructing machines to learn. We need to defeat the limits that characterize the four unique kinds of human-made consciousness, the boundaries that distinct machines from us – and us from them. When we talk about broad types of AI, we can generally see four AI system types. The four major types of AI are reactive AI, limited memory AI, theory of mind AI, and self-awareness AI. Let us briefly read about each of them.

a) Reactive AI: The most fundamental yet, at the same time, very valuable AI is called reactive machines since it responds to existing conditions as its name suggests. The Dark Blue, the supercomputer made by IBM, is a remarkable illustration of reactive AI.

Reactive AI works how it was modified with an anticipated output, dependent on the information it gets. These reactive AI machines will react to indistinguishable

circumstances in precisely the same manner without fail. There won't ever be a difference in real life. Reactive machines can't learn or imagine the past or future. Email spam filtering and Netflix suggestions are different examples of reactive AI.

b) Limited Memory: The following most complex AI is called limited memory AI. It's portrayed by the capacity to ingest learning information and improve over the long run, dependent on its experience, like how the human neurons are connected.

With limited memory, the AI is fabricated so that models are consequently prepared and afterward trained dependent on the model behavior.

Complicated classification problems can be pursued with limited memory AI. Self-driving vehicles utilize restricted memory AI since the AI that power these vehicles use the information they were prepared and modified to observe how to work. It can also understand available information it sees to peruse its current circumstance and change when in need.

c) Theory of Mind AI: When machines obtain dynamic abilities equivalent to people, we will have accomplished the Theory of Mind AI. A significant section of this artificial intelligence is that machines can comprehend and recall feelings and emotions and change their behavior depending on those feelings, similar to human beings' social, emotional connections.

The theory of mind will be better outfitted to work with people genuinely than different types of AI. Computer scientists are working passionately to find insights and gain ground, yet there's much work to do before AI can adequately treat every human emotion.

d) Self-awareness AI: The latest advancement of artificial intelligence is the improvement to construct frameworks that can shape portrayals about themselves (represents themselves). Finally, computer scientists have to build a smart system that can understand consciousness.

This AI hasn't grown effectively yet because we don't have the equipment or algorithms to uphold it. Up to that point, computer scientists will keep on upgrading restricted memory AI and creating a theory of mind AI.

We have seen the different dimensions of AI and the four significant kinds of AI. Let's have a look into the present-day usage of AI in the field of research, business applications, and day-to-day applications. As we have discussed, the learning phase of artificial intelligence is indeed essential. Thereby arises the need for a framework for learning the various aspects of the environment, human behavior, and working principle of different tasks. Intelligent machines can also perform on their own like humans, even for newly assigned tasks. Thus, AI gains another level of intelligence with the capacity of learning. With the invention of learning capacity, AI gains popularity with machine learning, where the machine can learn based on some training given to it. Gradually, machine learning needs upgradation too. Then few other domains arise, namely, deep learning and reinforcement learning. This book will cover each of these topics in detail in the upcoming chapters.

1.1.3. Why is Learning AI Important?

Artificial intelligence is significant because, generally, human abilities can be embraced in programming modestly and at scale. Artificial intelligence can be applied to each area to empower additional opportunities and efficiencies. The significance of artificial intelligence and machine learning can be seen through typical applications such as text summarizations, web crawlers, chatbots, personal assistants, and recommendation systems, which are a portion of the regular utilization of AI and ML. Indeed, even self-driving vehicles and meteorological offices give gigantic significance to AI and ML to foresee and forestall mishaps and disasters.

Artificial intelligence helped education vastly turn out to be more normal today, allowing educators and instructive organizations to evaluate kids and customize instruction philosophies to every kid. They can increment the productivity of organization responsibilities to permit instructors to influence more towards comprehension and versatility. By utilizing the best attributes of machines and educators, we can achieve the optimal result in AI and ML in the education sector.

With AI, technology enables human capabilities in thoughtfulness, reasoning, development, communication, and awareness. Artificial intelligence has various, unmistakable use cases today that empower corporate income development and cost reserve funds in existing areas. Applications will be generally various in areas where a considerable amount of time is spent gathering and orchestrating information, such as finance, retail, stock exchange, administrations, assembling,

and medical services. Utilizations of intelligent computer vision will be especially critical in the transportation sector.

Artificial intelligence is enormous because it effectively handles a significant arrangement of complex real-life problems.

Since its commencement during the 1950s, AI research has aimed at five fields of specialization:

1. *Knowledge or Information*: The capacity to address information about the world.
2. *Thinking Power or Capacity of Reasoning*: The capacity to tackle issues through sensible thinking. To reason is to apply rationale to determine convictions, related thoughts, and ends from data. Consideration might be deductive, inductive, or abductive.
3. *Planning for Proper Execution*: The capacity to set and accomplish objectives.
4. *Effective Way of Communication*: The capacity to comprehend composed and communicate in language. To speak with individuals, computers can recognize, comprehend, and combine spoken or written communication modes, especially in human language.
5. *Discernment or Perception of Derivation*: The capacity to make derivations about the world dependent on tangible information. The computer should have the option to arrange, recognize and decipher visual pictures, sounds, and other tangible data sources for better perception of knowledge.

1.2. NEED FOR MACHINE LEARNING

Machine learning is the subarea of artificial intelligence. Without programming the computer to act some task, machine learning helps the computer for self-learning. When taking care of new data, these machines learn, develop, change, and create without outside help. The idea of machine learning has been around for some time now. Nonetheless, the capacity for consequently and rapidly applying vast mathematical calculation to the colossal dataset is currently acquiring a touch of energy.

The arena of machine learning is continuously developing. Alongside the advancement comes ascent popularity and significance. One key motivation behind the fact is that computer scientists need to learn machine learning to advance in AI [3]. Machine learning is mainly comprised of three sections:

- The computational calculation is at the center of decision-making.
- Components and features that help in making a choice.
- Original data helps in learning the machine for further tasks.

Machine learning is now able to improve results, matching the precision of medical specialists. Directly, scientists feed the computer with a vast amount of data in the form of images and textual data to better understand the machine's ability to 'learn'. Different organizations dealing with machine learning for medical services, such as Google and Xerox, use many clinical images marked by specialists. Machine learning algorithms are released on these visual informational indexes, searching for factual examples to sort out what highlights of a picture, proposing its merits the specific mark.

1.3. WHAT IS LEARNING IN MACHINE LEARNING?

Machine learning is a section of artificial intelligence (AI) that gives frameworks the capacity to learn and improve for a fact without being customized naturally. ML centers around advancing programs that can get information and use it to find out on their own.

The way toward learning starts with some information, like models, direct insight, or guidance, to search for designs in information and settle on better choices later on depending on the models we give. The vital point is to permit the machines to adapt naturally without human intercession or help and change activities.

"Learning" is an indeed characterized term. Naturally, learning implies improving at performing on some duty with experience.

Tom M. Mitchell gave a broadly cited, formal definition:

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

Along these lines, the expression "learning" just truly has significance concerning some assignments.

Here are a few instances of an assignment:

Walking: children don't inherently have the foggiest idea of how to walk. However, instead, they endeavor to walk commonly. They acquire insight, and some learning calculation (in their cerebrum) learns the examples of neuron firings that lead to rare falling.

Speech: instances of human discussion encircle us, and we get it from the beginning in childhood improvement.

ML is seldom isolated from statistics. Machine learning utilizes the likelihood to adapt to the numerous sources of vulnerability as the calculation fits the model's parameters. A few models have stochasticity implicit in them, particularly on the off chance that they are utilized for displaying marvels that aren't deterministic regardless (for example, quantum mechanics, flipping a fair coin). Measurements likewise permit a model to reason about occasions it needs to represent yet can never notice (for example, rear ends). A third justification of ML to go to statistics is that we may have to reduce the measure of data. For instance, an ML algorithm for tackling visual performance may have to adapt to high accuracy, just as low image quality without decreasing the accuracy. It utilizes probability to counter the deficiency of data from the awful output to adjust coarse goal input images.

1.4. WHEN DO WE NEED MACHINE LEARNING?

By and large, machine learning is utilized when more restricted, structured information or data access. Most ML calculations are intended to prepare models for a structured form of data. If the information is unstructured, ML can be applied; however, it requires some data control techniques.

Deep learning typically requires enormous data or information to guarantee that the architecture might have many variables and doesn't overfit the model during training. Convolutional neural networks are intended to work on imaging datasets,

even though they can be utilized on sensor information by applying a time-frequency computation like a spectrogram. Recurrent neural networks like LSTM (long short-term memory) networks are intended to work on successive or sequential information like time series and textual data.

Machine learning is regularly utilized for projects that include predicting a result or uncovering patterns. In these models, a restricted assemblage of information is used to assist the machines with learning designs that they can later use to ensure new information. Standard algorithms utilized in ML incorporate linear regression, decision trees, support vector machines (SVM), naive Bayes, and linear discriminant analysis.

Deep learning models will set aside some effort during training. Several pre-trained networks and open-access datasets abbreviate the training process yet take a considerable time to execute. It also requires an ample amount of computational capacity of the computer. Computer scientists applying deep learning should invest a dominant part of their energy preparing models and adjust to their neural architecture model framework.

ML computations might be more attractive in the event when one needs faster outcomes. They are quick to prepare and require less computational force. The number of highlights and perceptions will be the key factors that influence training time. Computer scientists applying ML ought to invest a dominant part of their energy in creating and assessing highlights to improve model exactness.

Deep learning models will set aside some effort during training. Several pre-trained networks and open-access datasets abbreviate the training process yet take a considerable amount of time to execute. It also requires an ample amount of computational capacity of the computer. Computer scientists applying deep learning ought to invest a dominant part of their energy preparing models and adjust to their neural architecture model framework.

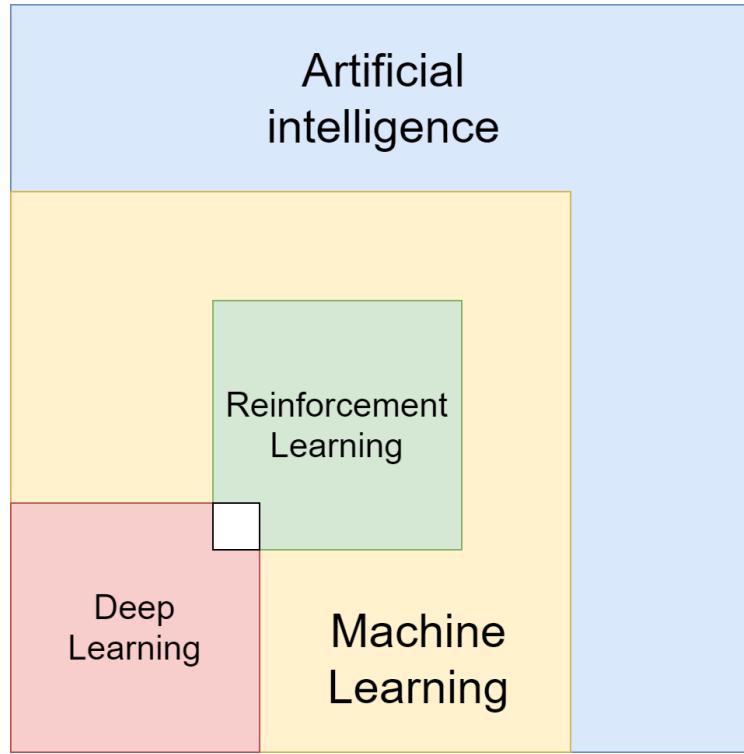


Fig. (1.4). Overview of the division of AI into its sub-fields - ML, DL and RL.

1.5. TYPES OF LEARNING

As discussed, learning is a very complex task in both humans and computers. It depends on various factors, domain, and maturity. Again, same with human learning and machine learning. In terms of computers, factors are the different parameters that affect the learning process, such as learning environment, features set, correlation of variables, dependent and independent variables, learning rate, and architecture. As for the domain, machine learning also depends on the data, such as image, signal, video, text, and other kinds of data. Finally, when we talk about maturity, same with the human learning process, the more we mature, the more we learn faster and precisely. Machine learning also needs the maturity of its learning model; that's why the pre-trained models are more efficient than the just-trained models. As shown in Fig. (1.4), it shows the overview of AI and its different subfields, such as machine learning, deep learning, and reinforcement learning.

Currently, there are basically three main types of learning available; supervised, semi-supervised, and unsupervised. However, recently, two more types of learning are getting popular among computer scientists; reinforcement learning and self-supervised learning. This book will talk about each of the types of learning in detail in the following chapters. The outline of the entire book is given at the end of this chapter to understand the book's flow. Here, we quickly visit each of the learning processes to get an overview of the same. As discussed, the five main learning algorithms are:

- a) Supervised learning
- b) Unsupervised learning
- c) Semi-supervised learning
- d) Reinforcement learning
- e) Self-supervised learning

1.5.1. Supervised

Supervised machine learning algorithms are used to predict future events using labeled examples to define what you previously learned from new data. Starting with an analysis of a set of available training data, the learning algorithm performs an approximate function to predict the output value. The system can provide output for new inputs after proper training. The learning algorithm can look for errors to compare the output to the exact intended output and correct the model accordingly.

1.5.2. Unsupervised

Unsupervised machine learning algorithms are primarily used for training, while the information used is not classified or labeled. Research into how systems determine the ability to explain hidden structures in unlabeled data arises from unsupervised studies. The system doesn't search for the correct output, but you can explore the data and search the dataset to explain the hidden structure in the data in the label.

1.5.3. Semi-supervised

Semi-supervised machine learning algorithms fall between supervised learning and unsupervised learning because they use both labeled and unlabeled data for training. Specifically, data with weak labels and data with few labels. Systems using this method can significantly improve learning accuracy. Semi-supervised learning is usually chosen when the acquired labeled data is skilled for training/learning and relevant resources. Otherwise, editing unlabeled data typically does not require additional resources.

1.5.4. Reinforcement

Reinforcement machine learning algorithms are learning methods that interact with the environment through tasks and detect errors or give rewards. Trial and error detection and delay compensation are the most relevant features of reinforcement learning. Using this method, one can maximize the efficiency of mechanical and software agents by automatically determining the ideal behavior in specific situations. Agents need simple reward feedback to know which action is best. This is known as reinforcement learning.

1.5.5. Self-supervised

Self-supervised learning is a method for preparing a computer to finish an assigned task without given named information, *i.e.*, labels. It is a part of unsupervised learning where outputs or objectives are determined by machines that classify or categorize and examine data independently, then make inferences dependent on associations and relationships.

Self-supervised learning can likewise be a self-governing type of regulated learning since it doesn't need human contribution to data classification. Rather than unsupervised learning, self-supervised learning doesn't focus on clustering and categorizing, which is generally familiar in unsupervised learning.

1.6. WHAT IS THE NEED FOR THIS BOOK ON MACHINE LEARNING?

The purpose of writing another book on machine learning is always a challenging task to attract the reader's attention. Machine learning is the most discussed topic of this decade and for a few more decades. The basic knowledge of machine

learning is very needed. Most people are unaware of the fundamental theories and applications of machine learning.

This book aims to cover most of the machine learning curriculum prescribed in most of the top universities. It will also cover advanced topics like deep learning and feature engineering. This book's added feature will be an entire chapter on real-world machine learning using python programming, which will be truly beneficial for all the researchers and engineers, with open-ended ideas on new problems and their solutions pythonic way.

This book is written effortlessly and straightforwardly with enriched theories and fewer mathematical complications but more easily comprehensive application aspects. In every chapter, topics are described in such a way, keeping in mind readers from all sections. Every topic and subtopic will be described with examples and python code snippets for a more accessible explanation. Chapters are presented with a well-explained illustration and flowchart for a better understanding of the topic. Thus, this book on machine learning will surely catch the beginners' attention in the machine learning domain. The audience will include university students, young researchers, ph.d. students, professors, and also the software engineers who want to gain knowledge in machine learning from scratch.

1.7. OUTLINE OF THE BOOK

This book dedicates all the chapters entirely to machine learning and deep learning architecture. As the name suggests, it deals with mostly machine learning applications and deep learning algorithms in real life. The overall outline of the book is given below:

Chapter 1 introduces the fundamental concepts of artificial intelligence and the gradual growth of artificial intelligence that gives rise to machine learning and deep learning. It throws a deep insight into the need for learning and the concept of learning in the machine learning framework. This chapter also states the importance of AI and its various dimensions. In the end, this chapter introduces machine learning and its variants. It also shortly described each type of machine learning algorithm.

Chapter 2 is entirely dedicated to supervised learning. It is the first chapter of Section I of this book. It starts with the introduction and theory of supervised learning and classification. This chapter describes in detail the overview and

algorithmic framework for the popular classification algorithms including decision tree, random forest, k-nearest neighbor, Naïve Bayes, and support vector machine algorithm. This chapter states the mathematical theory of each of these algorithms and describes with a hands-on example and working Python program.

Chapter 3 talks about the unsupervised machine learning algorithm, describing the state-of-the-art clustering algorithms. This chapter gives an elaborative definition of k-mean clustering, hierarchical clustering, and self-organizing map. It also defines the algorithmic framework of each algorithm with hands-on examples with detailed Python codes and outputs.

Chapter 4 describes the regression models. This chapter mainly focused on two essential regression algorithms, namely, linear regression and logistic regression. This chapter also describes each of them in detail supporting Python programs better to understand the regression process in a real dataset.

Chapter 5 describes reinforcement learning (RL), a machine learning subfield similar to supervised or unsupervised learning but differs in numerous aspects. It is more difficult to apply to real-world business scenarios since it requires simulated data and surroundings. On the other hand, RL technology is promising since its learning process is inherent to sequential decision-making settings. This chapter clearly explained the algorithm and its usage.

Chapter 6 starts with Section II of the book that describes the advanced machine learning part, *i.e.*, the deep learning approaches. This chapter is entirely devoted to deep learning architectures, starting from artificial neural networks to convolution neural networks until recurrent neural networks. This book also introduces the reinforcement learning concept to the readers. All the algorithms are supported with hands-on examples and Python code for better understanding. Detailed mathematical explanations with easy descriptions are given for all the mentioned algorithms in this chapter.

Chapter 7 deals with feature engineering. As we know, feature engineering is a pivotal part of machine learning technology, which helps identify the critical features of data. This chapter covers two sub-chapters, namely, filter-based feature selection and wrapper-based feature selection. This chapter discusses statistical feature selection techniques, such as t-test, ANOVA, and Pearson's correlation coefficient in the filter-based approach. This book talks about the popular

evolutionary algorithms in wrapper-based approaches, such as genetic algorithms, particle swarm algorithms, and other feature selection algorithms. Each algorithm is carefully described for easy understanding for readers from varied backgrounds with pictorial representations and algorithmic framework.

Chapter 8 is entirely dedicated to the application section of the book. The title of the book suggests more emphasis on the applicability of machine learning. This chapter is written for all the beginners in the machine learning and deep learning subject. This chapter presents the Python programs for various state-of-the-art problem areas. These problem areas are chosen carefully to cater to the variety of needs of every learner covering maximal domain-specific tasks. This chapter covers the domain of pattern recognition, including face recognition, object recognition, optical character recognition; The area of medical imaging and computational linguistics, in terms of natural language processing, are also presented carefully with detailed Python implementation on the real-life dataset. In the end, this chapter throws light on the probable most recent research problems, which need the immediate attention of computer scientists and AI specialists. This part will benefit the authors with the knowledge of current trends in AI research.

Chapter 9 finally concludes the book by summarizing all the chapters for a quick overview and future work.

CONCLUDING REMARKS

The introductory chapter introduces the fundamental concepts of artificial intelligence and the gradual growth of artificial intelligence that gives rise to machine learning and deep learning. It throws deep insights into the need for learning and the concept of learning in the machine learning framework. This chapter also states the importance of AI and its various dimensions. In the end, this chapter introduces machine learning and its variants. It also shortly described each type of machine learning algorithm.

CHAPTER 2

Supervised Machine Learning: Classification

Abstract: This chapter introduces supervised machine learning algorithms. In this chapter, the popular classification algorithms such as decision tree, random forest, k-nearest neighbor, Naïve Bayes classifier, and support vector machine are described in detail. Each algorithm is defined starting with its overview, followed by an algorithmic framework and a hands-on example. A detailed Python program is given at the end of each algorithm to support the precise understanding of the working behavior of the classifiers. The Python code is executed on a real dataset, which eventually gives the reader in-depth knowledge about the algorithm's applicability.

Keywords: Classifier, Decision Tree, Machine Learning, Random Forest, K-nearest Neighbor, Naïve Bayes Classification, Supervised Learning, Support Vector Machine.

2.1. INTRODUCTION TO SUPERVISED MACHINE LEARNING

2.1.1. Supervised Machine Learning

Supervised machine learning algorithms are used to predict future events using labeled examples to define what you previously learned from new data. Starting with an analysis of available training data, the learning algorithm performs an approximate function to predict the output value. The system can provide output for new inputs after proper training. The learning algorithm can look for errors to compare the output to the exact intended output and correct the model accordingly.

The training dataset is used to explain the model to get the desired product through supervised learning. This training dataset contains inputs and appropriate outputs so that the model can train over time. The algorithm calculates the accuracy with a loss function by adjusting the loss to a minimum value.

Supervised models can be used to build and enhance multiple business applications, including the following businesses:

1. Image and Object Identification: Supervised learning algorithms can be used to detect, distinguish, and classify objects in video or images, which can be applied to various computer vision techniques and image analysis.

2. Predictive Analytics: A broad use case of supervised learning models lies in developing predictive analytics systems that provide in-depth insights into various business data points. This enables entrepreneurs to expect specific outcomes based on given output variables, allowing business leaders to make decisions or give critical support to the company's benefit.

3. Customer Sentiment Analysis: With supervised machine learning algorithms, any company can extract and classify much information, including context, sentiment, and motivation, with minimal human intervention. This can be very useful to understand customer interactions better and can be used to improve one's brand investment trial.

4. Spam Detection: Spam detection is another instance of a supervised machine learning model. Using supervised classification algorithms, organizations can effectively organize spam and non-spam-related communications by training their databases to identify patterns or inconsistencies in new data.

Deep data insights and improved automation can benefit your supervised learning business, but building a sustainable supervised learning model presents some challenges. Some of these challenges are:

- Supervised learning models may require some degree of expertise to be accurately designed.
- Early model training through learning can be time-consuming.
- The data set is more likely to have human errors, resulting in algorithmic learning errors.
- Supervised learning cannot cluster or classify data on its own.

2.1.2. What is Classification?

In machine learning, classification refers to a predictive modeling problem in which class labels are predicted for a given example of input data. Classification is the task of using a machine learning algorithm that learns what class labels should be for instances in the problem domain. Categorizing it as "spam" or "no-spam" makes the example easier to understand.

A variety of classification features can give everyone a unique approach to machine learning and modeling used.

Classification algorithms are used to assign test data to specific categories accurately. It tries to draw conclusions on identifying particular elements in a data set and labeling or defining those elements. Standard classification algorithms are linear classification decision tree, random forest, k-nearest neighbor, and support vector machine (SVM) described in more detail below.

Examples of classification problems include:

- Classification of emails as to whether it is spam or not.
- Given a handwritten letter, classify it as one of the known letters.

Classification requires training and a database to learn a lot of information. The model takes a set of training data and calculates how to map instances of the input data at a specific class label. By the way, the training dataset should be sufficiently representative of the problem, and each class label should have multiple instances.

Class labels are often String values. "Spam", "No-Spam", and numerical values need to be mapped before providing the modeling algorithm. This is often mentioned as label encoding, and each class label is assigned a unique integer. "Spam" = 0, "No-Spam" = 1.

There are several types of classification algorithms that model problems to predict modeling classification.

They have no fixed solution on how to map the algorithm to the problem type. Instead, it is generally recommended that practitioners conduct controlled experiments and find the best algorithm and configuration for a given classification task.

The classification prediction algorithm is evaluated based on the outcomes. Classification accuracy is a standard system of measurement used to assess model performance based on predicted labels. The classification accuracy is not perfect, but it is a good starting point for many sorting tasks.

Instead of class labels, some operations may require an estimate of the class member potential for each instance. This provides additional uncertainty in the

forecast that can be explained later by the application or by the user. A widely used diagnosis to assess future potential is the receiver operating characteristic (ROC) curve.

2.1.3. Types of Classification

Basically, there are four core types of classification tasks available:

1. Binary classification
2. Multi-class classification
3. Multi-label classification
4. Imbalanced classification

1. Binary Classification: It denotes those classification tasks with a two-class problem, having two class labels. In short, the binary classification function has one class that is in the normal state and one that is abnormal.

2. Multi-class Classification: It mentions those classification tasks with three or more class labels, multi-group problems. Unlike binary classification, multi-class classification is unaware of normal and abnormal results. Instead, examples of one of the known class categories are classified. The number of class labels for some problems can be huge. For example, a facial recognition system can be estimated to have thousands of faces based on model estimates.

3. Multi-label Classification: It refers to those problems with more than two labels or classes. In this problem, each sample may be assigned multiple labels. Let us look at an example of photo classification. The picture can contain various objects in the scene, and the model picture can predict multiple known entities such as "Vehicle", "Fruit," and "Human".

4. Imbalanced Classification: This represents a classification function with an unevenly distributed number of examples in each class. In short, an unbalanced classification function is a binary classification function where most of the examples in the training data set are one class, and the other remaining examples are second class, which is less in number.

2.2. DECISION TREE

2.2.1. Overview

In the preceding section, we learned about supervised machine learning and its classification. This section will learn about the Decision Tree and its different types of algorithms that generally fall under supervised learning. Under the umbrella of supervised machine learning, the decision tree is one of the major topics to be covered. The decision tree is very similar to a flowchart and is mainly used as a tool for classification and prediction [4]. As the name suggests decision tree has a "tree" like structure. It has two primary attributes, one is the node, and the other is the leaf. Nodes denote a test on an attribute, and all single leaf holds a class label. The branch of the decision tree presents the output test. Fig. (2.1) illustrates a standard decision tree with its root node and branches.

Decision trees are sometimes denoted as Classification and Regression Trees (CART) since they are used in two contexts, regression, and classification.

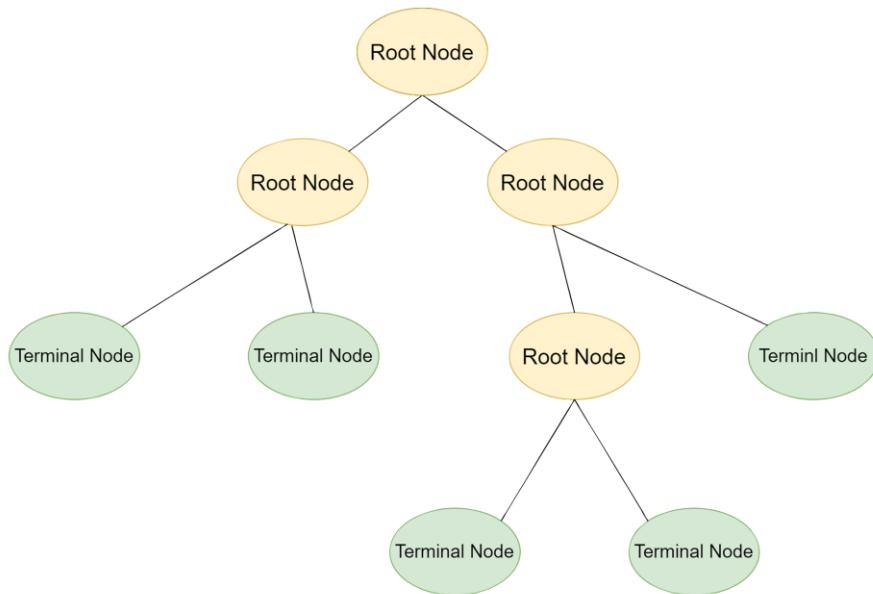


Fig. (2.1). Decision tree algorithm.

2.2.2. Algorithmic Framework

As mentioned before, a decision tree or CART is used in two different contexts, regression, and classification. This tree-like structure has some particular terminologies (like nodes, leaves, branches, *etc.*), which are essential to be discussed before we practice making decision trees. Remember, any boolean function on discrete attributes could be represented on the decision tree.

2.2.2.1. *Different Terminologies used in Decision Tree*

- **Root Node:** It is the starting point of any decision tree. According to the dataset, the root node is further divided into two or more homogeneous sets. It is the parent node.
- **Leaf Node:** The final outputs are represented as leaf nodes. There is no further division after a leaf node. Leaf nodes are also known as child nodes.
- **Splitting:** Division of root node or decision node into sub-nodes according to different conditions is splitting.
- **Branch or Sub Tree:** A tree formed by splitting the tree.
- **Pruning:** Removal of any different or non-wanted branches from the tree is known as pruning.

Along with some terminologies, there are some assumptions which we need to remember before we draw any decision tree. Some of the significant assumptions are mentioned below:

- **Assumption 1:** Initially, the root node is considered as the complete training.
- **Assumption 2:** Categorization of the feature values is done. Remember, continuous values are converted before building the model.
- **Assumption 3:** Distribution of the records is done according to the attribute values.
- **Assumption 4:** We use statistical methods for ordering attributes as root or the internal node.

The decision tree is divided into various levels. Each level needs different attributes. Identifying additional attributes at each level is the most challenging task while we form a decision tree. The process of finding different attributes is known as attribute selection. There are two main methods for attribute selection. One is information gain, and another is the Gini index.

2.2.2.2. Entropy

Measuring the uncertainty of a random variable is known as entropy. The value of entropy and the purity of information content are directly propositional, higher entropy more valuable content [4].

The generic formula of entropy can be drafted as:

$$\text{Entropy } (H) = - \sum_{i=1}^c P(x_i) \log_b(x_i)$$

Let's take an example to see how we can find out entropy:

Suppose, there is any set $S = \{x, x, x, x, y, y, y, y, y, y\}$

Number of total instances = 10

Number of instances of x = 4

Number of instances of y = 6

So, the entropy will be:

$$\text{Entropy } H(S) = -[\frac{4}{10} \log_2 \left(\frac{4}{10}\right) + \frac{6}{10} \log_2 \left(\frac{6}{10}\right)]$$

2.2.2.2.1. Purpose of Entropy

Entropy plays a significant role while we draw a decision tree. With the help of entropy, we can decide the splitting branches and make the boundaries of a decision tree. The value of entropy ranges between 0 to 1; the less the value of entropy more trustable it is.

2.2.2.3. Information Gain

There are chances that a decision tree formed could go under a transformation, which means the entropy of the decision tree will change. The change in entropy is calculated by information gain. Information gain calculates the difference between the "before and after entropy" of a decision tree after the transformation. With the help of information gain, we split the node and built the decision tree.

The algorithm of all the decision trees is always designed to increase the information gain's value. According to the rules of decision tree formation, the node or attribute with the maximum information gain value will be the first one to be split. The formula for finding the information gain is as follows:

$$\text{Information Gain} = \text{Entropy}(S) - [(\text{Weighted Average}) * \text{Entropy(each feature)}]$$

Entropy is a system of measurement to quantify the impurity in a given attribute. It postulates randomness in data. Entropy can be calculated as:

$$\text{Entropy}(S) = - P(\text{TRUE}) \log_2 P(\text{TRUE}) - P(\text{FALSE}) \log_2 P(\text{FALSE})$$

Where,

- S = Total number of instances
- P(yes) = probability of TRUE
- P(no) = probability of FALSE

Information gain also measures the change in the entropy observed in the decision tree due to the formation of the sub-sets, according to the given conditions.

2.2.2.4. Gini Index

When we choose an element, there is a probability that the selected element could be identified incorrectly. Thus, to measure this probability, we have the Gini index. It is a metric-like structure—the lower the Gini index value, the more preference a randomly chosen element is given.

The formula to calculate the Gini index is as follows:

$$\text{Gini Index} = 1 - \sum_j P_j^2$$

2.2.3. Hands-on Example

2.2.3.1. Building a Decision Tree With the Help of Information Gain

2.2.3.1.1. Requirements

- A. Initially, we will focus on the training terminology provided with the root node.
- B. Information label node for retrieving information gain. Attributes cannot be repeated.
- C. Create a summary of each subset of the subsets of training events that will be categorized along the training tree path.

2.2.3.1.2. Boundary Case

- A. If all positive or negative training examples remain, label the nodes "TRUE" or "FALSE" accordingly.
- B. In the absence of an attribute, label the node with the most votes among the remaining training events.
- C. If no instances remain, label the node with the most votes of the parent's training instance.

2.2.3.1.3. Basketball Data

The table given below (Table 2.1) shows sample data for a hypothetical basketball match.

Table 2.1. Basketball data.

Where	When	Fred Starts	Joe Offense	Joe Defense	Opposition	Outcome
Home	7pm	Yes	Center	Forward	Tall	Won
Home	7pm	Yes	Forward	Center	Short	Won

(Table 2.1) cont.....

Away	7pm	Yes	Forward	Forward	Tall	Won
Home	5pm	No	Forward	Center	Tall	Lost
Away	9pm	Yes	Forward	Forward	Short	Lost
Away	7pm	No	Center	Forward	Tall	Won
Home	7pm	No	Forward	Center	Tall	Lost
Home	7pm	Yes	Center	Center	Tall	Won
Away	7pm	Yes	Center	Center	Short	Won
Home	9pm	No	Forward	Center	Short	Lost
...

What we know:

- We know that the game will be played 'away' at 9 pm, and 'Joe' will play center on the offense side (Table 2.2).
- It is a classification side, as seen in Fig. (2.2), where we can see how the observations are derived based on the decision.
- Generalizing the learned rule to new examples.

Table 2.2. Test case to check decision tree.

Where	When	Fred starts	Joe offense	Joe defense	Opposition	Outcome
Away	9pm	No	Center	Forward	Tall	?

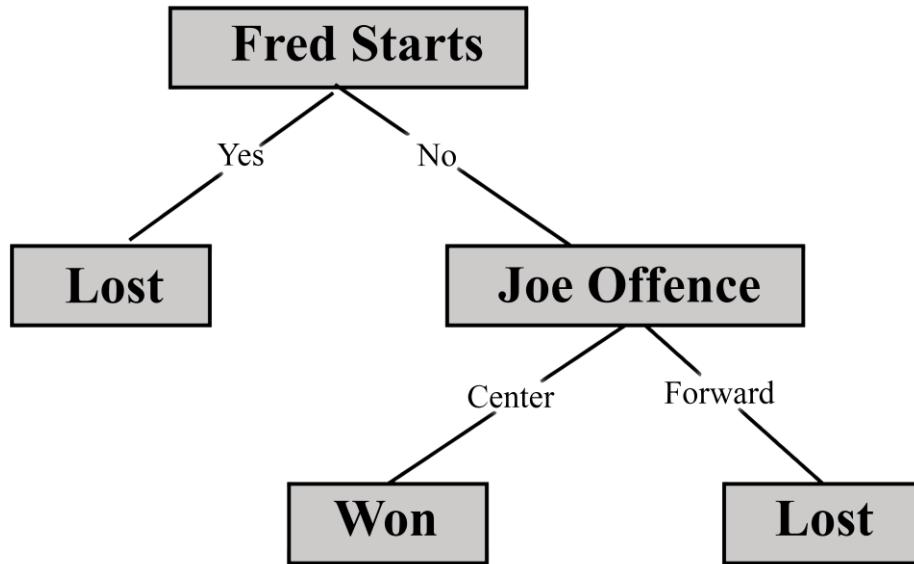


Fig. (2.2). Decision tree built on Table 2.1 data (intermediate level).

We have to raise three questions:

- i. Which node to start? (For identifying the root)
- ii. Which node to proceed with?
- iii. When to stop? (For identifying the terminal node)

Suppose S has 25 instances, 15 positives and 10 negatives [15 (true class), 10 (false class)]. Then the entropy of S corresponding to this classification is:

$$E(S) = -\left(\frac{15}{25}\right)\log_2\left(\frac{15}{25}\right) - \left(\frac{10}{25}\right)\log_2\left(\frac{10}{25}\right)$$

It should be noted that if the result is inevitable, then the entropy is '0'. If you don't know about the system, the entropy is maximal (or any result is equally possible).

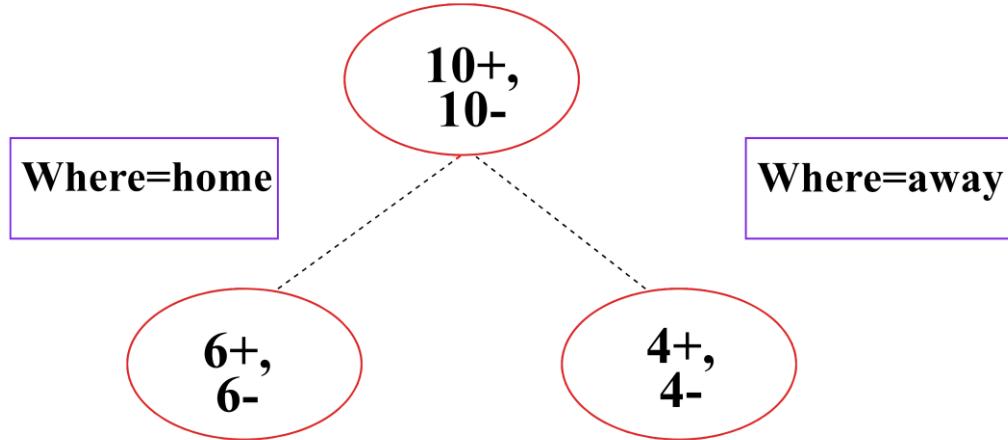


Fig. (2.3). Intermediate step showing the building of the tree.

- Before partitioning,

The entropy is $H(10/20, 10/20) = -10/20 \log(10/20) - 10/20 \log(10/20) = 1$

When we use the ‘where’ attribute, it divides into 2 subsets, as seen in Fig. (2.3).

$$\text{Entropy}_{\text{1st Set}} H(\text{home}) = E(S) = -\left(\frac{6}{12}\right) \log_2 \left(\frac{6}{12}\right) - \left(\frac{6}{12}\right) \log_2 \left(\frac{6}{12}\right) = 1$$

$$\text{Entropy}_{\text{2nd Set}} H(\text{away}) = E(S) = -\left(\frac{4}{8}\right) \log_2 \left(\frac{4}{8}\right) - \left(\frac{4}{8}\right) \log_2 \left(\frac{4}{8}\right) = 1$$

- Expected entropy after partitioning,

$$\frac{12}{20} * H(\text{home}) + \frac{8}{12} * H(\text{away}) = 1$$

2.2.3.1.4. In the Next Level

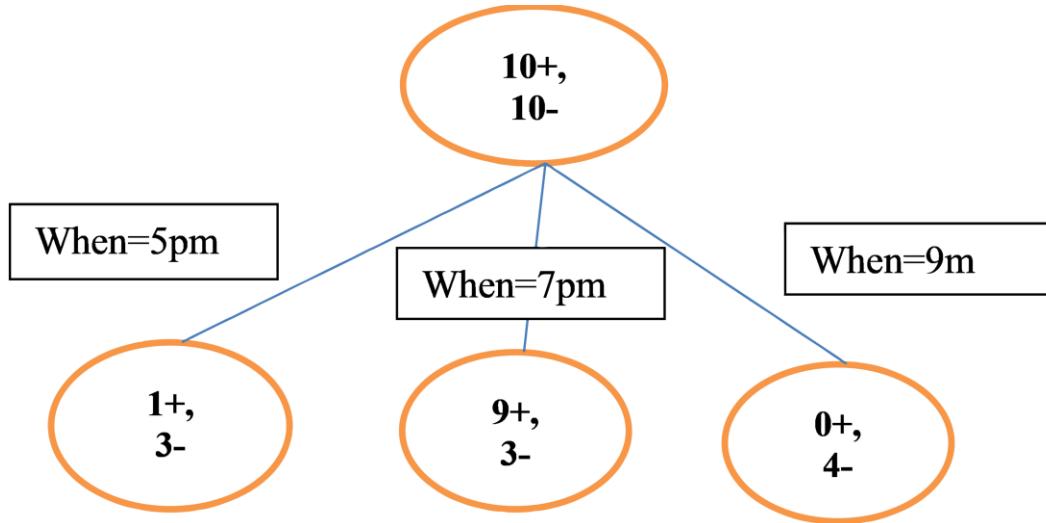


Fig. (2.4). The intermediate step of building tree while using "When" clause.

We use the 'when' attribute, it divides the tree into three subsets, as seen in Fig. (2.4):

$$\text{Entropy}_{\text{1st Set}} H(5\text{pm}) = E(S) = - \left(\frac{1}{4} \right) \log_2 \left(\frac{1}{4} \right) - \left(\frac{3}{4} \right) \log_2 \left(\frac{3}{4} \right) = 0.811$$

$$\text{Entropy}_{\text{2nd Set}} H(7\text{pm}) = E(S) = - \left(\frac{9}{12} \right) \log_2 \left(\frac{9}{12} \right) - \left(\frac{3}{12} \right) \log_2 \left(\frac{3}{12} \right) = 0.811$$

$$\text{Entropy}_{\text{3rd Set}} H(9\text{pm}) = E(S) = - \left(\frac{0}{4} \right) \log_2 \left(\frac{0}{4} \right) - \left(\frac{4}{4} \right) \log_2 \left(\frac{4}{4} \right) = 0$$

- Expected entropy after partitioning,

$$\frac{4}{20} * H\left(\frac{1}{4}, \frac{3}{4}\right) + \frac{12}{20} * H\left(\frac{9}{12}, \frac{3}{12}\right) + \frac{4}{20} * H\left(\frac{0}{4}, \frac{4}{4}\right) = 0.65$$

So, the information gain is 0.35 (1 - 0.65).

2.2.3.1.5. Decision

Here, we see that the value of information gain is more than the 'where' attribute using the 'when' attribute. Thus, considering the 'when' attribute is better than taking the 'where' attribute. Similar way, we should calculate the information gain of other attributes. If it is a non-terminating node, we should choose the attribute having a higher value of information gain.

2.2.4. Types of Decision Tree Algorithms

Before we build a decision tree, some decision tree algorithms are used foremost, and we should discuss them.

1. Iterative Dichotomiser 3 (ID3): It uses information gain to expand the decision tree. With the help of information gain, each attribute is assigned to a subset in the tree. In this algorithm, Information gain is calculated at each level of the tree for the remaining data.

2. C4.5: After ID3, C4.5 is the most used decision tree algorithm. This algorithm can use information gain or gain index to classify all the attributes and expand the tree. This algorithm can handle both continuous and missing attribute values together.

3. Classification and Regression Tree (CART): A dynamic learning algorithm relies on a regression tree and a classification tree based on variables.

2.2.5. Advantages and Disadvantages of the Decision Tree

2.2.5.1. Advantages

- Generates of understandable rules
- Performs classification, less computation needed
- Can handle continuous and categorical variables
- Provides the most accurate prediction or classification

2.2.5.2. Disadvantages

- Limited in the prediction of the value of a continuous attribute

- When training examples are fewer, and many classes are available, then decision trees are prone to errors.
- Expensive training.

2.2.6. Programming Approach for Decision Tree

```
Decision tree classifier implementation

import numpy as np

from matplotlib import pyplot as plt

# Dataset relevant imports

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris


# Decision tree relevant importss

from sklearn.tree import DecisionTreeClassifier

from sklearn import tree


# For calculating accuracy and creating confusion matrix

from sklearn.metrics import confusion_matrix

from sklearn.metrics import accuracy_score

from sklearn.metrics import classification_report


# Load the dataset

iris = load_iris()

X = iris.data
```

```

y = iris.target

# Split the dataset into training and test

X_train, X_test, y_train, y_test = train_test_split(X, y, random_
state=0)

# Instantiate the classifier

decision_tree = DecisionTreeClassifier(criterion = "gini", max_le
af_nodes=3, random_state=0)

# Train the classifier

decision_tree.fit(X_train, y_train)

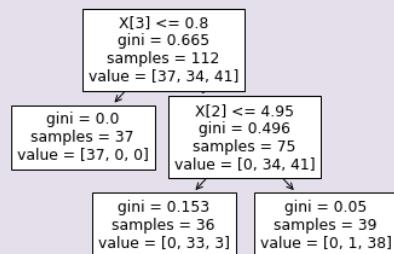
# Display decision Tree

tree.plot_tree(decision_tree)

plt.show()

```

OUTPUT:



```
# Make predictions

y_predicted = decision_tree.predict(X_test)

print("Predicted values:")

print(y_predicted)

OUTPUT:
Predicted values:

[2 1 0 2 0 2 0 1 1 2 1 1 1 0 1 1 0 0 1 1 0 0 1 0 0 1 1 0 2 1 0
1 2 1 0 2]

# Print Confusion Matrix

print(confusion_matrix(y_test,y_predicted))

print("-----")

# Print Accuracy Score

print(accuracy_score(y_test, y_predicted)*100)

OUTPUT:
[[13  0  0]
 [ 0 15  1]
 [ 0   3  6]]

-----
89.47368421052632
```

2.3. RANDOM FOREST

2.3.1. Overview

Regarding the supervised machine learning technology, random forest is one of the most popular and used algorithms. Random Forest is a classifier with numerous decision trees over multiple subsets of a given dataset and takes averages to improve the prediction accuracy of that dataset. Random forest is an algorithm that could be used for classification and regression problems of machine learning. They

improve the overall model because they can solve multiple complex problems faster and accurately.

Random forest algorithm is more accurate because the solution is not based on just one single tree. It is based on the number of trees. Take estimates from all available trees and make the final product based on the highest part obtained. Having many trees in the forest increases accuracy and prevents overfitting problems.

2.3.2. Why Use Random Forest?

There are several advantages for using random forest, below are some specific points:

- Less training time.
- Highly accurate output.
- Efficient run.
- Accurate results even with a miss portion of the data.

2.3.3. Algorithmic Framework

Combining an infinite number of trees forms a forest in nature. Similarly, the N number of decision trees creates a random forest in supervised machine learning. The random forest algorithm works in two different phases; firstly, it combines the N number of decision trees and then calculates each tree's prediction in the forest (Fig. 2.5).

Read the below steps to understand the complete working process of a random forest algorithm:

Step-1: From the available training set, random data sets are selected.

Step-2: From the chosen data points, we now build a decision tree.

Step-3: Choose all the decision trees that need to be made.

Step-4: Repeat the Step 1 and 2.

Step 5: For new data points, find the predictions for each decision tree and assign the latest data points to the category with the most votes. The function of the algorithm can be better understood with the following example.

Example: Suppose you have a dataset containing multiple fruit images. So, this dataset is fed into a random forest classifier. The data is divided into subsets, and each decision is provided in a tree. In the training phase, each decision tree produces a prediction result. Based on most of the products, the random forest classifier predicts the final decision as new data points arrive. Consider the following image.

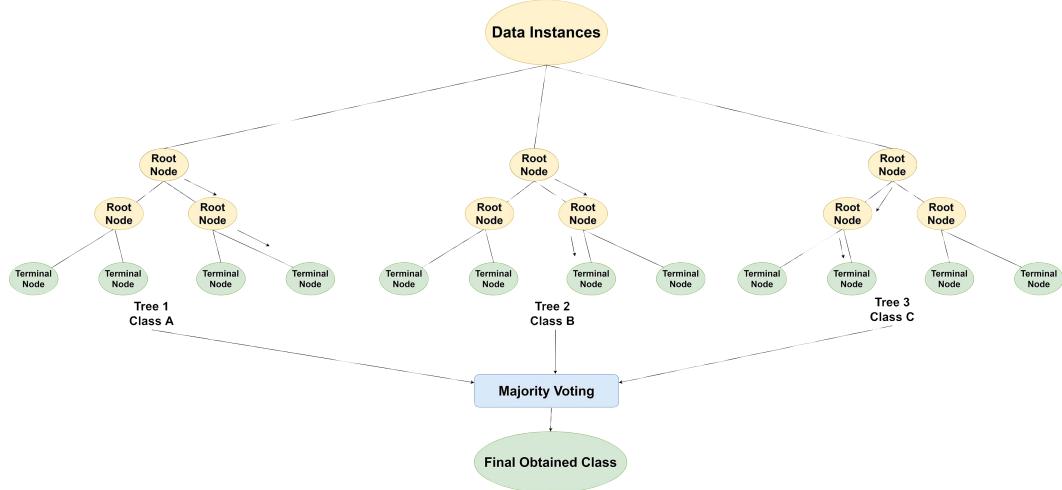


Fig. (2.5). Random forest algorithm.

2.3.3.1. Assumptions for Random Forest

When compiling an output from N number of decision trees, different decision trees predict different outputs. Some outputs could be correct and incorrect, but collectively, the random forest indicates the most accurate output. Therefore, two assumptions should be followed for an accurate random forest prediction:

- i. Character variables in the data set must have some tangible value so that the classification can predict the exact outcome instead of the expected result.
- ii. Estimates for each plant should have very low correlations.

2.3.4. Advantages and Disadvantages of Random Forest

2.3.4.1. Advantages

- Can perform random forest classification and regression functions.
- Random Forest can handle large data sets with high dimensions.
- This increases the accuracy of the model and avoids the best problems.

2.3.4.2. Disadvantages

Random forest algorithm may be used for both classification and regression tasks, but it is less suitable for regression operations.

▪ Random Forest is used Primarily in Four Sectors of our Society

1. Banking: Mostly used in the identification of loan risks.
2. Medicine: Identification of the severity of the diseases.
3. Land Use: Identification of similar lands.
4. Marketing: Identification of the ongoing trends.

2.3.5. Programming Approach for Random Forest

```
Implementation of random forest

# Imports for the dataset

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris


# Imports for random forest classifier

from sklearn.ensemble import RandomForestClassifier


# For evaluating the classifier
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Load example dataset

iris = load_iris()

X = iris.data

y = iris.target

# Split the dataset into training set and test dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

random_forest_clf = RandomForestClassifier(max_depth=2, random_state=0)

random_forest_clf.fit(X_train, y_train)

# Predict the labels of the testdataset

y_predicted = random_forest_clf.predict(X_test)

print("Predicted values:")

print(y_predicted)
```

OUTPUT:

Predicted values:

```
[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0
2 2 1 0

2]
```

```
# Print Confusion Matrix  
  
print(confusion_matrix(y_test,y_predicted ))  
  
print("-----")  
  
# Print Accuracy Score  
  
print(accuracy_score(y_test, y_predicted)*100)
```

OUTPUT:

```
[[13  0  0]  
 [ 0 15  1]  
 [ 0  0  9]]
```

```
-----  
97.36842105263158
```

2.4. K-NEAREST NEIGHBOR

2.4.1. Overview

In 1951, Evelyn Fix and Joseph Hodges developed a non-parametric classification algorithm known as K-Nearest Neighbor (KNN).

KNN algorithm could be used for classification and regression problems. In the KNN algorithm, the input should have a 'k' nearest training example, but the output depends entirely on the problem type.

The output of a classification problem is always a class member. Objects are sorted by multiple neighbors that are assigned to the most common category of objects among the closest neighbors (K is a positive integer, usually tiny). If k = 1, the object is only served to the nearest adjacent square.

If you have a regression problem, the output object has a property value. This value is the average of the nearest neighbor values. KNN is designed to predict new data using information from the nearest K neighbor from existing data.

KNN doesn't have a process that can be called learning. That's because when new data comes in, only neighbors are selected by measuring the distance between the existing data. That's why some people call KNN a lazy model, meaning that they don't build a model separately. It is also called Instance-based learning a little politely. It is a concept that contrasts with Model-based learning, which creates a model from data and performs a task, and is intended to perform tasks such as classification/regression using only each observation without a separate model creation process [5].

The hyperparameter of KNN, *i.e.*, K , used a way to measure two distances. If K is small, the data model is more valuable. Conversely, if it is too large, the model is underfitting. The size of K simplifies the scope of the classification.

However, it should be noted that the KNN algorithm does not directly create these interfaces, and it visualizes which categories are classified when given new data.

2.4.2. When Do We Use KNN Algorithm?

We now know that KNN could be used for classification and regression problems. Still, it is more likely to be used for classification problems on a larger scale. KNN is evaluated on three main aspects:

1. Easy interpretation of output
2. Time to calculate the output
3. Power to predict the accurate output

2.4.2.1. Distance Indicator

KNN is an algorithm whose results vary greatly depending on the distance measurement method. Let's look at a few.

▪ Euclidean Distance

This is the most commonly used distance measure—the shortest straight-line distance between two observations.

$$d_{(x,y)} = \sqrt{\sum_{i=1}^n (x_i - y_i)^2} \text{ for all } i = 1, 2, 3, \dots, n$$

For example, in Fig. (2.6), given below, we see a straight line joining two points A and B:

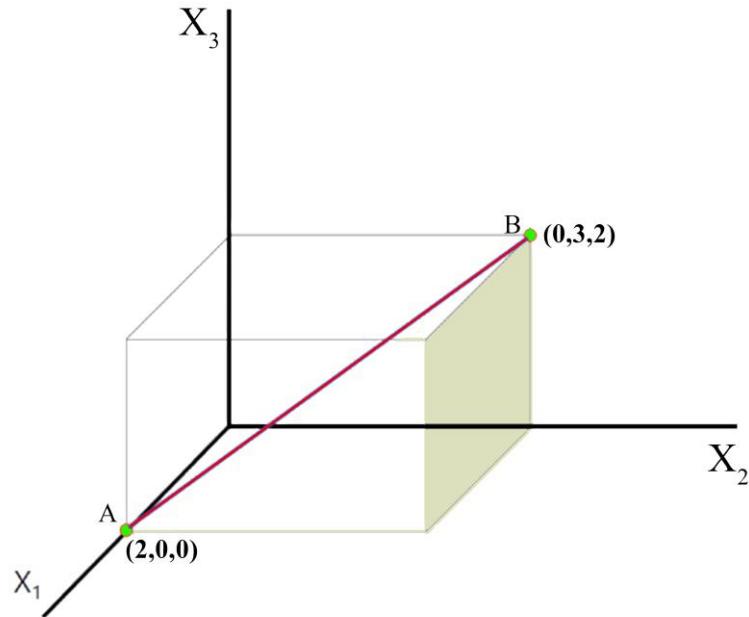


Fig. (2.6). Two points (A and B) in the \$X_1\$, \$X_2\$ and \$X_3\$ axis.

$$d_{(x,y)} = \sqrt{\sum_{i=1}^3 (0 - 2)^2 + (3 - 0)^2 + (2 - 0)^2} = \sqrt{17}$$

▪ Manhattan Distance

Distance calculated when moving from A to B only in the direction of each axis. To get from one building to another in Manhattan, New York, you must follow a lattice-shaped path, which is easy to remember—also called Taxicab Distance.

$$d_{(x,y)} = \sum_{i=1}^n |x_i - y_i|$$

▪ Mahalanobis Distance

This method calculates the distance by reflecting both the variance within the variable and the covariance between the variables. This is a distance indicator that considers the correlation between variables.

$$d_{Mahalanobis(x,y)} = \sqrt{(\vec{x} - \vec{y})^T \Sigma^{-1} (\vec{x} - \vec{y})}$$

Σ^{-1} = Inverse of Covariance matrix

If you set the Mahalanobis distance between x and y as c, x as x_1, x_2 , and y as (0,0) and solve the above equation, you can write as follows. It is the form of the equation of an ellipse.

$$x_1^2 s_1 + 2x_1 x_2 s_2 + x_2^2 s_4 = c^2$$

$$\Sigma^{-1} = \begin{bmatrix} s_1 & s_2 \\ s_3 & s_4 \end{bmatrix}$$

If the covariance matrix becomes an identity matrix, then Mahalanobis distance is equivalent to Euclidean distance. To measure distance, Mahalanobis distance considers both the difference in the distance variable and the adjustment or correlation between the variables.

▪ Correlation Distance

Use the Pearson correlation of the data directly as a measure of distance. It is a measure for comparing trends across data rather than individual observations. In

other words, it can reflect the similarity of the two data patterns. The correlation coefficient's value ranges from -1 to 1, so the correlation distance ranges from 0 to 2. If it is 0, you can interpret the pattern of the two data as very similar, and if it is 2, it is not.

$$d_{corr(x,y)} = 1 - \rho_{xy}$$

▪ Rank Correlation Distance

Use the Spearman Rank correlation of the data directly as a measure of distance. The remaining features are the same as the correlation distance.

$$d_{corr(x,y)} = 1 - \rho_{xy}$$

$$\rho_{xy} = 1 - \frac{6}{n(n^2 - 1)} \sum_{i=1}^n (rank(x_i) - rank(y_i))^2$$

Suppose, for example, that the seasonal temperature ranks for each region are given as follows, as seen in Table 2.3:

Table 2.3. Seasonal temperature rank in three metropolitan city.

Area	Spring	Summer	Autumn	Winter
Seoul	3	1	2	4
New York	3	1	2	4
Sydney	2	4	3	1

So, then, the rank correlation distance between Seoul and New York is as follows:

$$d_{corr(Seoul,New\ York)} = 1 - \rho_{xy} = 1 - 1 = 0$$

$$\rho_{Seoul, New York} = 1 - \frac{6}{4(4^2-1)} \{(3-3)^2 + (1-1)^2 + (2-2)^2 + (4-4)^2\} = 1$$

2.4.3. How to Select the Value of K?

The best K is data specific, so you need to find it greedily. The K value in the KNN algorithm plays an important role; thus, the selection of the K value must be made precisely.

Following are some pointers to be considered while we choose the K value for the algorithm:

- The first step for choosing a particular K value is Hit and Trial. You must place different values to get one of the most accurate values among them. 5 is the most preferred K value.
- Low values like 1 or 2 are preferred less because they create noise in the final output and affect model accuracy.
- Large values for K are found to be suitable, but some difficulties could be observed in some cases.

In addition, to find the best K, we divide the training data and the verification data; one must experiment while changing the value of K.

2.4.4. Algorithmic Framework

2.4.4.1. How Does K-NN Work?

The following algorithm describes the KNN function.

- Step 1: Choose a K value for your neighbor.
- Step 2: Calculate the Euclidean distance for K neighbors.
- Step 3: Get the nearest neighbor K according to the calculated Euclidean distance.
- Step 4: Count the number of data points in each of these K neighbors.
- Step 5: Allocate a new data point in the range with the maximum number of neighbors.
- Step 6: The model is ready.

2.4.4.2. Pseudo Code of KNN

The implementation of the KNN model is described in the next step:

- Step 1: Load data
- Step 2: K values are given.
- Step 3: To get an approximate rating, iterations should run from 1 until the last number of loaded points.
 - i. Measure the distance between each row of test data and training data. Here, Euclidean distance is the most widely used method, so we use distance as the metric. Other metrics you can use are Chebeshev, cosine, etc.
 - ii. Sort the calculated distance in ascending order based on the distance value to get top K rows from the sorted array
 - iii. Find the most frequent square of this row
 - iv. Return of expected class

2.4.5. Programming Approach for K-Nearest Neighbor

```
Implementation of k-nearest neighbor classifier

# Imports for dataset

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris


# Imports K-Nearest Neighbour Classifier

from sklearn.neighbors import KNeighborsClassifier


# For evaluating the classifier
```

```
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score

# Load example dataset

iris = load_iris()

X = iris.data

y = iris.target

# Split the dataset into training and test dataset

X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=0)

# Instantiate the classifier from sklearn

knn_clf = KNeighborsClassifier(n_neighbors=3)

# Train the Model

knn_clf.fit(X_train, y_train)

# Predict labels on the test data

y_predicted = knn_clf.predict(X_test)

print("Predicted values:")

print(y_predicted)

OUTPUT:
PREDICTED VALUES:

[2 1 0 2 0 2 0 1 1 1 2 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0
2 2 1 0 2]

# Print Confusion Matrix
```

```

print(confusion_matrix(y_test,y_predicted))

print("-----")

# Print Accuracy Score

print(accuracy_score(y_test, y_predicted)*100)

OUTPUT:

[[13  0  0]
 [ 0 15  1]
 [ 0  0  9]]

-----
97.36842105263158

```

2.5. NAÏVE BAYES CLASSIFIER

2.5.1. Overview

Naïve Bayes classifier is a classification method based on Bayes' Theorem. The assumption of this classification is the independence among the predictors. Thus, the Naïve Bayes classifier assumes that the availability of one feature in the class is not affected by different features in the class.

For example, a ball could be a tennis ball only if it is green in color, about 6.7 cm, and weighs around 55 grams. Together all these three properties make a ball a tennis ball irrespective of other available features.

Building this classification model is not just easy but works effectively on enormous data sets. It is simple to handle and outruns all the previous models of classification.

The bias theorem provides a way to measure the probability of the answer $P(c|x)$ at $P(c)$, $P(x)$, and $P(x|c)$. See the equation below:

$$P(c|x) = \frac{P(x|c)P(c)}{P(x)} P(c|X)$$

$$P(x_1|c) \times P(x_2|c) \times P(x_3|c) \times \dots \times P(x_n|c) \times P(c)$$

Where,

$P(c|x)$ stands for the posterior probability of class (c), with predictor (x).

$P(c)$ stands for the prior probability of class.

$P(x|c)$ stands for the likelihood.

$P(x)$ stands for the prior probability of predictor.

2.5.1.1. Conditional Probability Model of Classification

Have you wondered how flowers bearing plants are classified? It is done with different colors, sizes, and the number of flowers, but machine learning is done through predictive modeling. We have read and understood different classification algorithms, but the conditional probability model of classification is a method that helps us predict the output by various observations. The conditional probability model classification algorithm is used for problems where we don't need a numerical output like regression problems. In such problems, the input is said to be X, and the output is said to be Y.

Together, X and Y represent observations collected in the domain. A table or matrix of training data (columns and rows or features and samples) is used to fit the model. Models should learn to map specific examples to class labels or $y = f(X)$ with minimal misclassification errors.

A method to resolve this issue is to build a probabilistic model. From a probability point of view, we are interested in estimating the conditional probability of a class label given an observation.

For example, in a classification problem, k class labels y_1, y_2, \dots, y_k and n input variables, x_1, x_2, \dots, x_n . One can compute the conditional probability for a class label using a given instance or set of input values for x_n .

$$P(y_i|x_1, x_2, \dots, x_n)$$

You can then compute the conditional probability for each class label, and the highest probability label is likely to be returned as a classification.

Conditional probabilities are impractical, but they can be computed as joint probabilities. The Bayes theorems give you a fundamental way to calculate conditional probabilities.

The simplest form of calculation for the Bayes theorem is:

$$P(A|B) = P(B|A) * P(A)/P(B)$$

The probability we are interested in computing $P(A|B)$ is called the latter probability, and the $P(A)$ event is called marginal probability.

You can use Bayes' theorem to organize classification into conditional classification problems.

$$P(y_i|x_1, x_2, \dots, x_n) = P(x_1, x_2, \dots, x_n|y_i) \times P(y_i)/P(x_1, x_2, \dots, x_n)$$

It is easy to guess from the previous $P(y_i)$ data set, but conditional observation probabilities based on the class $P(x_1, x_2, \dots, x_n|y_i)$ are not possible unless the number of instances is too large. It is large enough to effectively predict the probability distribution for all possible combinations of values.

Similarly, applying the Bayes theorem directly also becomes impractical when the number of variables or features increases.

2.5.1.2. Calculating the Prior and Conditional Probabilities

Now we focus on how to calculate the elements of the equation used in the Naïve Bayes algorithm. The previous $P(y_i)$ is estimated by dividing the training dataset's observation frequency with the label class as the total number of examples in the training dataset.

The conditional probability for the value of a feature given a class label can also be estimated from the data—precisely, one data distribution per variable and examples of data belonging to a given class. If one has K classes and n variables, one must create and maintain $k \times n$ different probability distributions.

Different types of data for each feature require different approaches. Specifically, the data are used to estimate the parameters of one of the three standard probability distributions.

For categorical variables like labels, one can use a multinomial distribution. If the variable is binary, such as yes/no or true/false, you can use a binomial distribution. Gaussian distribution is often used when the variable is as numeric as the measure.

- Binary Variable: Binomial distribution.
- Categorical Variable: Multinomial distribution.
- Numeric Variable: Gaussian distribution.

The three distributions are very popular that they are often given the name Naive Bias implementation. Such as:

- Binomial Naive Bayes
- Multinomial Naive Bayes
- Gaussian Naive Bayes

Datasets with mixed data types for the input variables may need to select a different kind of data distribution for each variable.

It is not necessary to use one of these three normal distributions. For example, if a real-valued variable has another kind of distribution, for example, exponential, one can use that specific distribution. In some instances, a real-valued variable doesn't have a properly defined distribution, like bimodal or multimodal. It is suggested to use a kernel density estimator instead to estimate the probability distribution.

Naive Bayes algorithms have proven to be effective and are therefore widely used in text classification functions. Words in the document can be encoded as binary (with terms), numeric (word events), or frequency (TF/IDF) input vectors and binary, polynomial, or Gaussian probability distributions, respectively.

2.5.2. Hands-on Example

2.5.2.1. Make Predictions with Naive Bayes

We have understood how the Naïve Bayes algorithm works with the concept of probability and probability distributions. Let us predict an example dataset (as

observed in Table 2.4) for a football match with three attributes: Field as home ground and away; Team as domestic and international; and Outcome or the target class as Win and Defeat.

Table 2.4. Football match data 1.

Field	Team	Outcome
Home	Domestic	Win
Away	International	Win
Home	Domestic	Win
Home	Domestic	Win
Home	Domestic	Win
Away	International	Defeat
Away	International	Defeat
Home	Domestic	Defeat
Home	International	Defeat
Away	International	Defeat

Let us convert some categorical values to a number. One input of each instance carries two values, and the target class also has two values. So, converting each categorical values into binary, we get:

- 1) Variable:
 - a) Field
 - i) Home = 1
 - ii) Away = 0
 - 1) Variable:
 - a) Team
 - i) Domestic = 1
 - ii) International = 0

1) Variable:

- a) Outcome
 - i) Win = 1
 - ii) Defeat = 0

Therefore, we can restate the dataset as seen in Table 2.5:

Table 2.5. Football match data 2.

Field	Team	Outcome
1	1	1
0	0	1
1	1	1
1	1	1
1	1	1
0	0	0
0	0	0
1	1	0
1	0	0
0	0	0

Let's try Naïve Bayes Theorem to predict,

$$P(h|d) = (P(d|h) * P(h))/P(d)$$

Where:

- $P(h|d)$ = probability of hypothesis 'h' for the given data d, which is also known as posterior probability.
- $P(d|h)$ = probability of data 'd' for the given hypothesis 'h', when it was true.
- $P(h)$ = probability of hypothesis 'h', when it is true, which is also known as the prior probability of h.
- $P(d)$ = probability of the data.

In fact, we don't need the possibility to predict a class for new data values. The only thing required is the numerator and the most extensive response class, predicting the target class.

$$\text{MAP}(h) = \max(P(d|h) * P(h))$$

First, let's take the records from the dataset and use the trained model to tell which class they belong to.

weather = Home, car = Domestic

For the potential of the model, we connect both classes and compute the response. The output begins with a "Win" response. Multiply the conditional probabilities together and multiply the event probabilities of the class.

$$\text{Win} = P(\text{Field}=\text{Home} | \text{Output}=\text{Win}) * P(\text{Team}=\text{Domestic} | \text{Output}=\text{Win}) * P(\text{Output}=\text{Win})$$

$$\text{Win} = 0.8 * 0.8 * 0.5 = 0.32$$

Let us use the exact computation to find the 'Defeat' class:

$$\text{Defeat} = P(\text{Field}=\text{Home} | \text{Output}=\text{Defeat}) * P(\text{Team}=\text{Domestic} | \text{Output}=\text{Defeat}) * P(\text{Output}=\text{Defeat})$$

$$\text{Defeat} = 0.4 * 0.2 * 0.5 = 0.04$$

We see that 0.32 is greater than 0.04, so we can easily predict the output class as 'Win'.

2.5.3. Advantages and Disadvantages of Naïve Bayes

2.5.3.1. Advantages

1. It's simple, fast, and accurate, even for multi-class predictions.
2. The Naïve Bayes classifier compares well with other models such as logistic regression, requiring less training data.

3. This algorithm performs better for categorical variables than for numeric variables. For numerical variables, it assumes a normal distribution.

2.5.3.2. *Disadvantages*

1. If a categorical variable cannot be found in the training dataset, the model will assign a zero probability and cannot predict. This is frequently mentioned as "zero frequency". To solve this, one can use a smoothing technique. The modest smoothing method is Laplace estimation.
2. On the other hand, Naïve Bayes is also known as a lousy predictor, so the output probability is not considered very seriously.
3. Another limitation of Naïve Bayes is the understanding of the independent predictors. It is almost impossible to get a set of entirely different predictions in real life.

2.5.4. **Tips for Using Naive Bayes Algorithm**

This section throws few important practical tips when working with Naive Bayes models:

1. KDE for Complex Deployment

Suppose the potential distribution of a variable is complex or unknown. In that case, it is best to use a kernel density estimator or KDE to estimate the distribution directly from the data sample.

2. Performance Degradation Due to Continuity Change

Naive Bayes, by definition, assumes that the input variables are independent of each other. Some or most of the variables depend on reality, but this often works well. However, the efficiency of the algorithm decreases because it depends more on the input variable.

3. Prevent Numeric Underflow with Log

To compute separate conditional probabilities for instances of class labels, you need to combine multiple possibilities, one for each class and one for each input variable.

Likewise, multiples of minimal numbers can become mathematically unstable, particularly as the number of input variables surges. To solve this issue, it is easy to change the sum of the probabilities calculated from the product of probabilities.

For example:

$$P(y|x_1, x_2, \dots, x_n) = \log(P(x_1|y)) + \log(P(x_2|y)) + \log(P(x_n|y))$$

Computing the natural logarithm of the probability produces a large (negative) number and adds the numbers so that the large probability approaches zero. Often the resulting values can be compared and maximized to provide a class label. This is often referred to as the logarithmic trick when multiplying probabilities.

4. Update the Probability Distribution

When new data becomes accessible, it can be relatively straightforward to use this fresh data and the old data to update the parameter estimates for each change probability distribution. This allows the model to easily access new data or an ever-changing distribution of data over time.

5. Use as a Generation Model

The probability distribution summarizes the conditional probability of each input variable value for each category label. These probabilities can be helpful beyond what is commonly used in distribution classification models. For example, you can randomly sample the potential distribution to create new instances of the possible data. The assumed conditional free assumption could mean that examples can be given more or less based on the actual interdependencies between the input variables in the data set.

2.5.5. Programming Approach for Naïve Bayes Classifier

```
Implementation of naive bayes classifier

# Imports for dataset

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris
```

```
# Import NBClassifier - Gaussian
from sklearn.naive_bayes import GaussianNB

# For evaluating the classifier
from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score

# Load example dataset
iris = load_iris()

X = iris.data
y = iris.target

# Split the dataset into training and test dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, random_
state=0)

# Instantiate the classifier from sklearn
naive_bayes_clf = GaussianNB()

# Train the Model
naive_bayes_clf.fit(X_train, y_train)

# Predict labels on the test data
```

```
y_predicted = naive_bayes_clf.predict(X_test)

print("Predicted values:")
print(y_predicted)

OUTPUT:

Predicted values:

[2 1 0 2 0 2 0 1 1 1 2 1 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0
2 2 1 0 1]

# Print Confusion Matrix

print(confusion_matrix(y_test,y_predicted))

print("-----")

# Print Accuracy Score

print(accuracy_score(y_test, y_predicted)*100)

OUTPUT:

[[13  0  0]
 [ 0 16  0]
 [ 0  0  9]]

-----
100.0
```

2.6. SUPPORT VECTOR MACHINE

2.6.1. Overview

"Support Vector Machine" (SVM) is an algorithm that could be used for both types of challenges, classification and regression. Still, it is preferred more for the classification model. When using the SVM algorithm, consider a single data point

in the N dimension when the value of each feature is the value of a specific coordinate. It then finds and classifies data points using the hyperplanes that make a massive difference between the two classes.

The support vectors are the data points which hold the hyperplane. The SVM classifier tries to build the best boundary for both classes (hyperplane or line). Fig. (2.7) shows the basic principle of the support vector machine. The charm of SVM is that it is a well-designed machine learning algorithm based on a stunning and solid theoretical background. The magic of the algorithm is that it is practical because it is easy to implement in several ways and has useful functions [6].

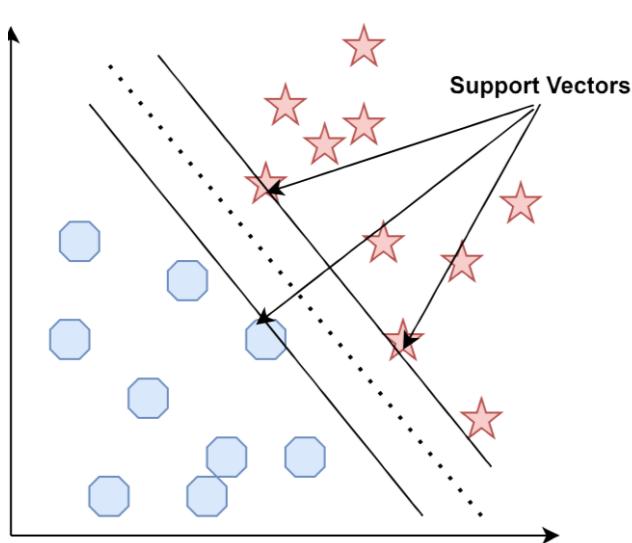


Fig. (2.7). Support vector machine.

The problems that need to be solved in SVM are when one needs to divide the dataspace using a decision boundary.

Let us take two sample inputs, '+' and '-', now question how to divide them into two classes (Fig. 2.8). Once we draw a line that divides both the data points equally, the easiest way is to draw a line to keep an equal and wide distance between the two data samples and the line. This line should maintain the most comprehensive possible distance with the data points. In Fig. (2.9) let us say that the drawn line is the dotted line, so the distance between the dotted line and the line passing through the closest data point should be broad.

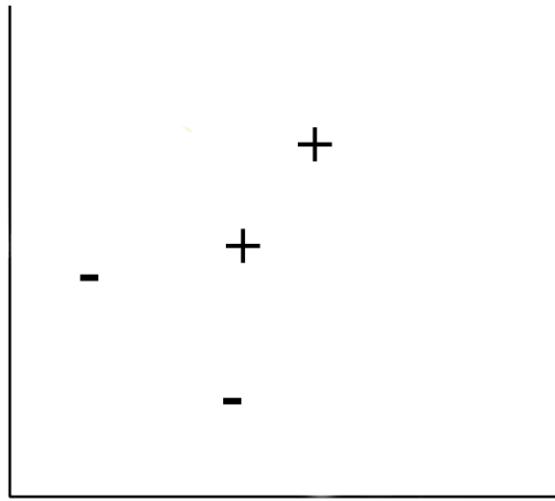


Fig. (2.8). Two kinds of data points (+ and -).

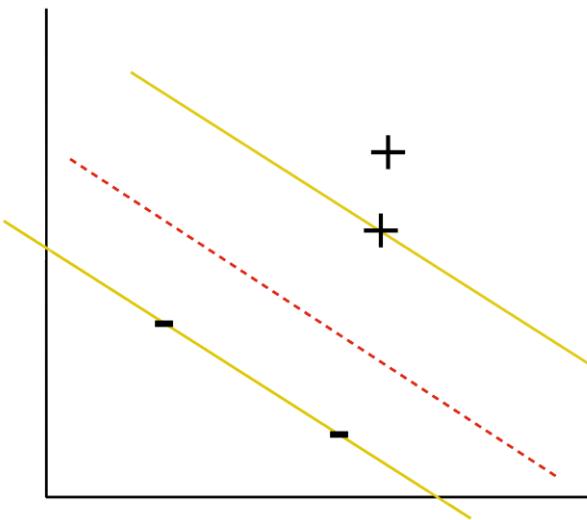


Fig. (2.9). The hyperplane trying to separate the points based on support vectors.

Even an intuitive solution seems to be a trivial problem. However, generalizing the answer to this question into a solid and logical theory is not very simple. SVM is a static method to find the solution to the desired problem.

2.6.1.1. Decision Rule

First, let us understand the nature of the decision rules determining the boundary's limits. Let's draw a vector \vec{w} , which is a vector that is orthogonal to the centerline.

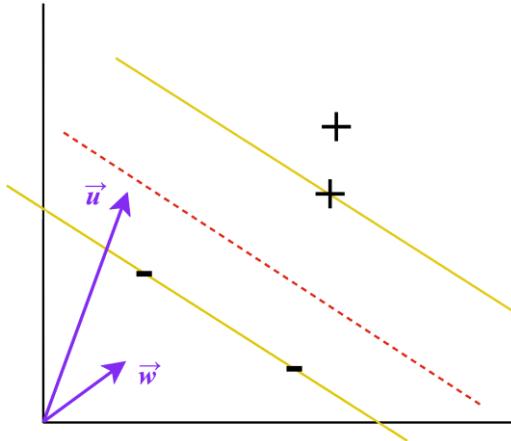


Fig. (2.10). Lines are drawn based on decision rule.

The most curious part is about when there's another vector \vec{u} , we don't know the position, whether on the right or left, based on position. So, one way we can do it here is to put in \vec{w} , and \vec{u} , and see if the value is more significant than any constant c using a dot product ($\vec{w} \cdot \vec{u} \geq c$).

$$\vec{w} \cdot \vec{u} + b \geq 0, \text{then the datapoint is '+ '}$$

The logic is straightforward. To do the inner mechanisms means to project \vec{u} above to \vec{w} . It is easy to understand if you think about its length, so it belongs to the right if you cross a boundary or to the left (Fig. 2.10).

So the formula mentioned above becomes the decision rule. It's also the first tool we need to understand SVM. But there are still many shortcomings. Yet we have no idea what \vec{w} should be determined, and how b should be determined in that formula. It's just one thing to know that \vec{w} is orthogonal to the centerline.

Unfortunately, such \vec{w} can be drawn in various ways, so the judgment here is that there is still a lack of constraints. So, we can add more constraints to see the variability.

2.6.2. Hands-on Example

2.6.2.1. Working of SVM

Till now, we are aware that a hyper-plane segregates two different classes. Now we will learn about how to find the correct hyper-plane.

Case-1: Identify the Right Hyper-plane: Suppose we have three hyper-plane names A, B, C, and you must find the star and circle hyper-plane (Fig. 2.11).

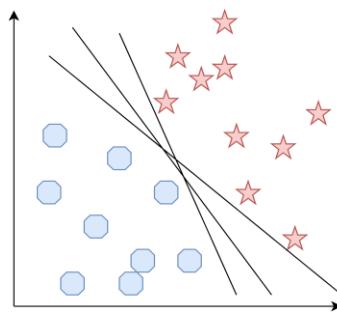


Fig. (2.11). Identifying the right hyperplane.

The most crucial rule to identify the correct hyper-plane is: "Correct hyper-plane is the one which segregates the two classes more accurately." Thus, in case-1, hyper-plane B is the most accurate one.

Case-2: Again, we have three hyper-plane (A, B, C).

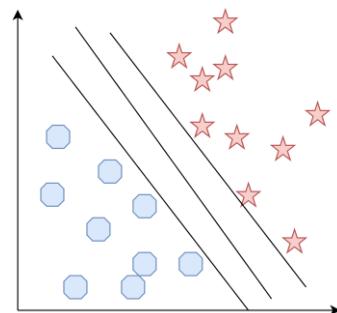


Fig. (2.12). Choosing the right hyperplane.

We look for the plane with the maximum distance from the nearest data point from any class (Fig. 2.12).

In the given data, hyper-plane C has the maximum distance; thus, it is the correct hyper-plane in this case. The distance between the hyper-plane and the nearest data point is called margin.

Case-3: Note: Use the rules outlined in the previous section to identify the correct hyperplane.

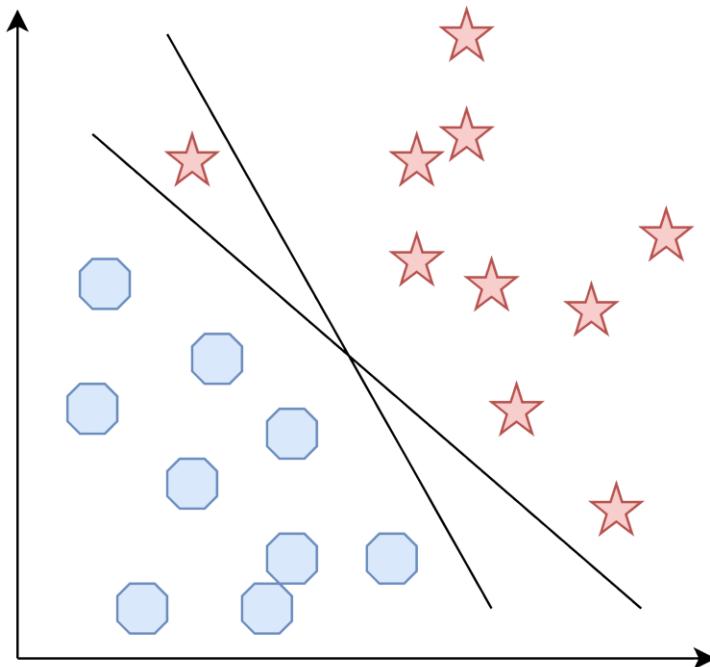


Fig. (2.13). Selecting correct hyperplane.

Some of you may have opted for Hyper-Plane B over A because it has high margins but high S (Fig. 2.13). Here, Hyper-Plane B has a classification error, and A is all correctly classified. So, the hyperplane is A.

Case-4: Can We Classify Two Classes? Below, you can't use a straight line to separate the two classes because one star is an outlet in the other (circle) class area (Fig. 2.14).

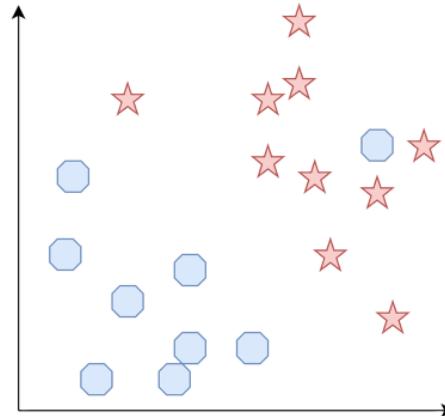


Fig. (2.14). Two classes, one data point being missed with other.

As mentioned earlier, the star at the other end is like the outer layer of a star class. The SVM algorithm can ignore the outlet and find the hyperplane with the maximum margin (Fig. 2.15). So, it can be said that the classification of SVM is vital for outlets.

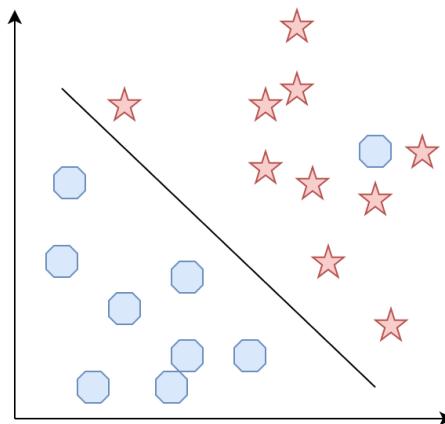


Fig. (2.15). Separating the missed classes with a hyperplane.

Case-5: Find the Hyper-plane to Segregate into Classes: We cannot have a hyperplane of the line between the two categories in the following situation, as in Fig. (2.16). So, how would SVM classify these two classes? So far, we have only looked at linear hyperplanes.

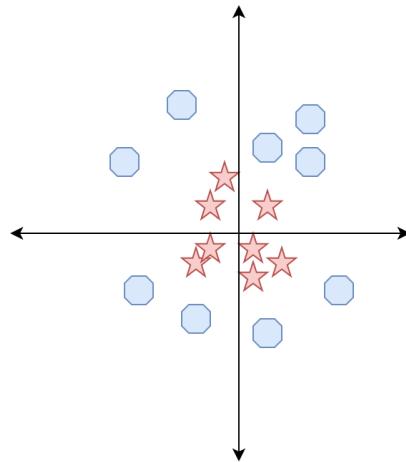


Fig. (2.16). Non-linear distribution of data.

SVM can solve this problem by introducing additional features. Here we add a new function to $z = x^2 + y^2$. Now plot the data points with z points on the axis and x .

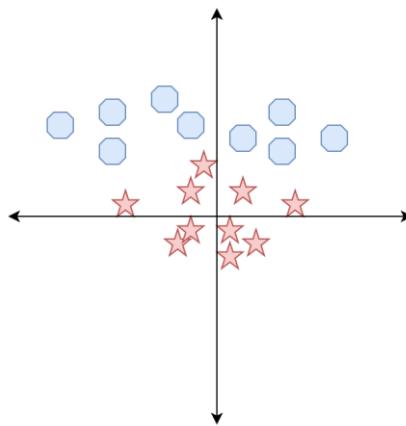


Fig. (2.17). Data distributing among mixed coordinates.

Consider the following from the Fig. (2.17), given above. Since x and y are square roots, all values of z are always positive. A red circle appears near the origin of the x and y -axis in the original figure, reducing the z value, and the star moves to a relatively higher z value than the actual result.

It's easy to have a linear hyperplane between these two classes in an SVM classifier. However, there may be another problem: do we have to add this feature to the hyperplane manually? No, the SVM algorithm has techniques like kernel tricks. The SVM kernel is a function that receives a low-dimensional input space and converts it to a high-dimensional space, transforming an indivisible problem into a partitioning problem. This is useful for most linear separation problems. In short, it does a very complex data transformation and then explores the process of separating the data based on labels or defined outputs. Looking at the hyperplane in the original input space, it seems like a circle (Fig. 2.18).

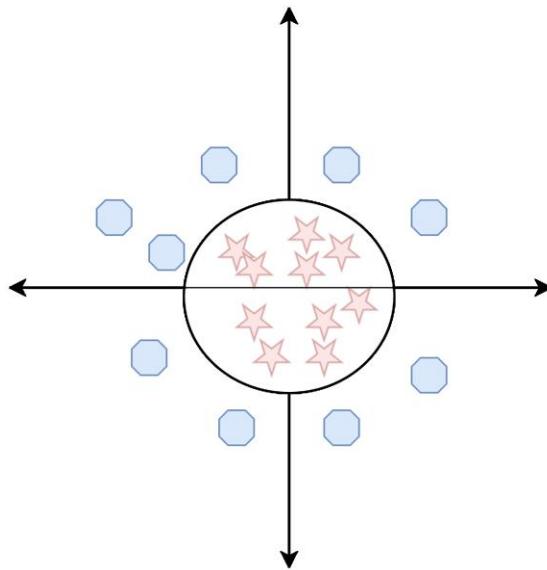


Fig. (2.18). Non-linear hyperplane.

2.6.3. The Kernel Trick

In this example, we found a clever mapping of large spaces to classify non-linear data. However, calculating these transformations is expensive on the computer. There can be many new dimensions, and each dimension can contain complex calculations. This can be much work for each vector in your data set. So, it would be great if you could find an inexpensive solution.

There is a trick. SVM doesn't need real vectors to do its magic; it can only be obtained with dot products. This means we can go beyond expensive math to a new level! Instead, it does the following:

Let us assume a new space:

$$z = x^2 + y^2$$

The dot product will be:

$$a \cdot b = xa \cdot xb + ya \cdot yb + za \cdot zb$$

$$a \cdot b = xa \cdot xb + ya \cdot yb + (xa^2 + ya^2) \cdot (xb^2 + yb^2)$$

Let us instruct SVM to do the job, but we call it a kernel function with a new dot product. This is a kernel trick that allows one to review costly calculations. Usually, the kernel is linear, and with it, linear classification is done. However, with a non-linear kernel, one can get a non-linear classifier. Changes in data change the dot product to the location you want, and the SVM will perform.

The kernel trick isn't part of the SVM. It can be used with other linear classifications such as logistic regression. The support vector machine only handles finding the boundaries.

2.6.3.1. Choosing a Kernel Function

When we are ready with the feature vector, one needs to select a learning model's kernel function. Each problem is different, and the kernel functions also depend on data. The data consisted of concentric circles in this example, so we chose a kernel that matches those data points. The three algorithms, namely, SVC, NuSVC, and LinearSVC, are helpful for binary and multi-class classification on a single data set. As seen in Fig. (2.19), there are found kernels working in the Iris dataset.

SVC and NuSVC are the same way, but they accept slightly different sets of parameters and have various mathematical calculations. On the other hand, you can implement lineRSVC support vector classification much faster for a linear kernel. The linearSVC parameter assumes it is linear, so it doesn't allow the kernel. Some features of SVC and NuSVC are missing, such as support. Like other classifiers, SVC, NuSVC and LinearSVC take two arrays as inputs. That is size array with training pattern, size array X, and size label array Y.

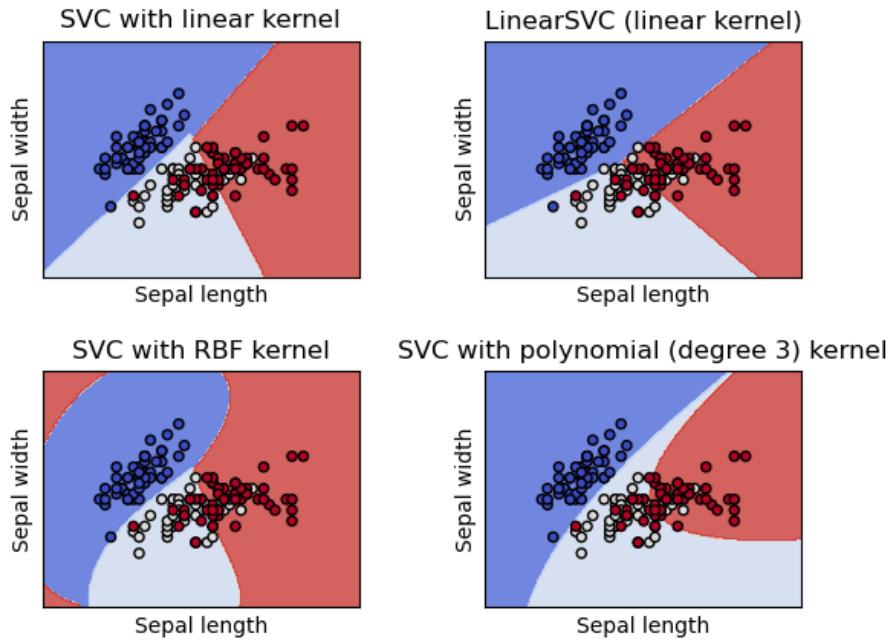


Fig. (2.19). Various kernels with Iris data.

If you use SVM in different areas, you can use hundreds of features. Meanwhile, the NLP classifier uses thousands of features to have at most one word for each word appearing in the training data. This changed the problem slightly. While using a non-linear kernel may be a good idea in other respects, many of these features make non-linear kernel data more useful. So it's better to stick to only good old linear kernels, in which case you can get good performance in these cases.

2.6.4. Advantages and Disadvantages of Support Vector Machines

2.6.4.1. Advantages

- Accurate in high-dimensional space, even with a large number of dimensions and a large number of samples.
- Efficient memory.
- Versatility: Various kernel features can be specified for decision-making. A generic kernel is provided, but you can also select a custom kernel.

2.6.4.2. Disadvantages

- a) If the number of features exceeds the number of samples, then the due normalization date is vital for choosing a kernel function to avoid overfitting.
- b) SVM does not directly predict probabilities and is computed using the expensive 5-fold cross-validation.

2.6.5. Programming Approach for Support Vector Machine

```
Implementation of support vector machine classifier

# Imports for the dataset

from sklearn.model_selection import train_test_split

from sklearn.datasets import load_iris


# Import SVM from sklearn

from sklearn import svm


# For evaluating the classifier

from sklearn.metrics import classification_report, confusion_matrix,
accuracy_score


# Load example dataset

iris = load_iris()

X = iris.data

y = iris.target


# Split the dataset into training and test
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_
state=0)

# Instantiate the classifier from sklearn

svm_clf = svm.SVC()

# Train the Model

svm_clf.fit(X_train, y_train)

# Predict labels on the test data

y_predicted = svm_clf.predict(X_test)

print("Predicted values:")

print(y_predicted)

OUTPUT:

Predicted values:

[2 1 0 2 0 2 0 1 1 1 2 1 1 1 0 1 1 0 0 2 1 0 0 2 0 0 1 1 0 2 1 0
2 2 1 0 2]

# Print Confusion Matrix

print(confusion_matrix(y_test,y_predicted))

print("-----")

# Print Accuracy Score

print(accuracy_score(y_test, y_predicted)*100)
```

```
OUTPUT:  
[[13  0  0]  
  
 [ 0 15  1]  
  
 [ 0  0  9]]  
-----  
97.36842105263158
```

CONCLUDING REMARKS

This chapter introduced supervised machine learning algorithms. This chapter explained the algorithms of popular trees such as Decision Trees, Random Forests, K-Neighbors, Name Bias Classifiers, and Support Vector Machines. Each algorithm began with an overview, was then described with an algorithm framework and hands-on examples. A detailed program for Python is provided at the end of each algorithm to understand a practical understanding of the functional behavior of the classification. The Python code was executed on real data sets and ultimately gives the reader an in-depth knowledge of algorithm applications.

CHAPTER 3

Unsupervised Machine Learning: Clustering

Abstract: This chapter introduces the readers to clustering algorithms as a part of unsupervised machine learning algorithms. This chapter describes the state-of-the-art clustering algorithms. This chapter gives an elaborative definition of k-mean clustering, hierarchical clustering, and self-organizing map. It also defines the algorithmic framework of each algorithm with hands-on examples with detailed Python codes and outputs.

Keywords: Machine Learning, Unsupervised Learning, Clustering, K-means Algorithm, Hierarchical Clustering, Self-organizing Map.

3.1. INTRODUCTION TO UNSUPERVISED MACHINE LEARNING

The unsupervised machine learning algorithms are primarily used for training, while the information used is not classified or labeled. Research into how systems determine the ability to explain hidden structures in unlabeled data arises from unsupervised studies. The system does not search for the correct output, but it can explore the data and search the dataspace to explain the hidden structure in the data and label them.

3.1.1. What is Clustering?

Back in the 1850s, John Snow, a London physician, developed a map where he plotted Cholera deaths. With the help of these maps, he identified that majority of deaths were reported near a well. Thus, this map exposed the problem and solution altogether.

Clustering is a fundamental task, where n numbers of data points are differentiated based on similarity and dissimilarity between them. From scattered data, smaller groups are formed, known as a cluster. The points in one cluster are similar to each other but different from the points in another cluster.

Clustering is a unique method that helps us identify the group among all the available scattered data. There is no fixed pattern or steps to perform clustering on the given set of data. The users are free to use their preferred clustering method

according to their needs. Clustering algorithms do make some assumptions depending on the provided data and make clusters accordingly [5].

3.1.2. Types of Clustering Methods

Several clustering methods could be used for different available data:

1. Density-based Methods: This is one of the best methods to merge two different clusters because such methods only consider a dense cluster with similar points. The different points are situated at a low, dense region in the same space. DBSCAN and OPTICS are some of their examples.

2. Hierarchical-based Methods: Forming a tree-like structure, the hierarchical-based method uses the previously formed data to form all the new clusters. There are two categories in this method:

- Agglomerative (bottom-up approach)
- Divisive (top-down approach)

CURE and BIRCH are some examples of hierarchical-based methods.

3. Partitioning Methods: Working on these methods is different from other methods. The available data is parted into k clusters in this method, and all the partitions form one cluster. This technique is used to optimize an objective function. Like k-Means, CLARANS is one example of partitioning methods.

4. Grid-based Methods: These methods are fast and independent because the entire data space is divided into a smaller finite number of cells like a grid. The clustering operation becomes very easy in such data space. Sting and clique are some examples of grid-based methods.

3.1.3. Real-life Applications of Clustering

- **Natural Disasters:** The study of disasters like earthquake and tsunami helps us determine dangerous zones and take different precautions.
- **Marketing:** It helps in discovering the customer segments for better marketing strategies.

- **Planning a City:** It helps in the identification of geographical locations for better housing plans and other infrastructure.
- **Insurance:** It helps in the identification of frauds.
- **Biology:** It helps in the identification and classification of different species of flora and fauna.
- **Libraries:** It helps in the classification of different books based on topics and information.

3.2. K-MEANS CLUSTERING

3.2.1. Overview

Unlike supervised learning, k-means clustering is an unsupervised learning algorithm. In k-mean clustering, we cluster a set of data into different clusters. Each cluster has similar data, which is closely arranged, whereas the dissimilar data is placed away from that particular cluster. The similarity and dissimilarity of the data are checked through nature and characteristics. The 'k' in the k-mean clustering stands for the number of clusters to be formed. There are various methods to find the optimum value of k. Let us see some real-life examples of k-mean clustering:

- Grouping articles (for example: google news)
- Grouping people with similar taste
- Identification of patients with high-risk health issues
- Classification of criminals from ordinary people

3.2.2. Algorithmic Framework

As discussed in the previous section, K-Mean is one of the simplest methods to solve clustering problems. Thus, now it is essential to understand the K-Mean Algorithm in depth.

3.2.2.1. *Introduction to K-means Algorithm*

In the K-Mean clustering algorithm, first, we calculate the optimum centroid for the given data. This is done by the iteration method. The first assumption here is that we know the K-value, *i.e.*, the number of clusters. This algorithm type is also

known as the flat clustering algorithm. K is the number of clusters calculated by the algorithm [6].

K-Mean clustering algorithm has been designed based on the distance between the data points and centroid. The placing of data in the cluster is done when the squared distance between the data point and centroid is the minimum. Thus, in other words, minor variation among the data point of a cluster, more similar to the data points.

3.2.2.2. Working of K-means Algorithm

The following steps will help us understand the algorithm of K-Mean clustering:

Step 1 – Initially, we have to generate the value of K by this algorithm.

Step 2 – Then, we move towards data classification depending on the number of data points.

Step 3 – The algorithm will compute the centroids of the cluster.

Step 4 – Keep on iterating for the optimum centroid. From which the distance between the data point and centroid is not changing.

First, the sum of the square distances between the data point and the centroid (center) is calculated. Now, let us allocate each data point to the closest cluster of the other. Finally, average all the data points in that cluster and calculate the centroid of the cluster.

K-means follows the **Expectation-Maximization** approach to solve the problem. The Expectation-step is used to assign the data points to the closest cluster, and the Maximization-step is used to compute each cluster's centroid.

There are some limitations when we use the K-mean clustering algorithm:

- Since K-mean clustering uses distance to form clusters, standardizing the data is done for better and relevant results.
- Allocating a random value of K could prevent the algorithm from reaching a global optimum. Thus, it is recommended to use different initializations of centroids.

3.2.2.3. Application of K-means Clustering Algorithm

- To generate a meaningful intuition from the data we are working on.
- Find clusters and then predict the different models that will be built for different subgroups.

K-mean clustering is helpful in various real-life platforms like:

- Clustering of documents
- Market segmentation
- Image segmentation
- Image compression
- Customer segmentation
- Analyzing the trend on dynamic data

3.2.3. Hands-on Example

The following two examples of implementing the K-Means algorithms:

- **Example with Hands-on Data Approach**

Let us take a simple example with a small dataset showing the implementation of k-means algorithm, using K=2 (follow the Table 3.1).

Table 3.1. Sample data set for k-means algorithm.

Individual	Var 1	Var 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0

(Table 3.1) cont.....

6	4.5	5.0
7	3.5	4.5

■ Step 1: Initialization:

- Here, we choose randomly two centroids ($k=2$) for two clusters. To start with, $M_1=(1.0,1.0)$ and $M_2=(5.0,7.0)$. We can see the change in Table 3.2.

Table 3.2. Data after first iteration.

Individual	Var 1	Var 2
1	1.0	1.0
2	1.5	2.0
3	3.0	4.0
4	5.0	7.0
5	3.5	5.0
6	4.5	5.0
7	3.5	4.5

Group 1: Datapoint number 1, Mean coordinate: (1.0, 1.0)

Group 2: Datapoint number 2, Mean coordinate: (5.0, 7.0)

■ Step 2:

- Now, we obtain two different clusters having data points {1,2,3} and {4,5,6,7}. After this, the new centroids are:

$$M_1 = (1/3(1.0+1.5+3.0), 1/3(1.0+2.0+4.0)) = (1.83, 2.33)$$

$$M_2 = (1/4(5.0+3.5+4.5+3.5), 1/4(7.0+5.0+5.0+4.5)) = (4.12, 5.38)$$

$$D(M1, 2) = \sqrt{|1.0 - 1.5|^2 + |1.0 - 2.0|^2} = 1.12$$

$$D(M2, 2) = \sqrt{|5.0 - 1.5|^2 + |7.0 - 2.0|^2} = 6.10$$

Now, we will have a new centroid table having two groups of datapoints as cluster, as follows in Table 3.3.

Table 3.3. After next iteration.

Individual	Centroid1	Centroid 2
1	0	7.21
2	1.12	6.10
3	3.61	3.61
4	7.20	0
5	4.72	2.5
6	5.31	2.06
7	4.30	2.92
"		

▪ Step 3:

- Now, using these centroids, we calculate the distance using Euclidean measure, as shown in table below. Thus, the new clusters are: {1,2} and {3,4,5,6,7}. Next centroids are: m1=(1.25,1.5) and m2 = (3.9,5.1), as seen in Table 3.4.

Table 3.4. Data after step 3.

Individual	Centroid1	Centroid 2
1	1.57	5.38
2	0.47	4.28

Table 3.4) cont.....

3	2.04	1.78
4	5.64	1.84
5	3.15	0.73
6	3.78	0.54
7	2.74	1.08

■Step 4:

- Now, the new clusters are {1,2} and {3,4,5,6,7}. Thus, we observe no change in the cluster in terms of the allocation of data points (see Table 3.5).
- Thus, the algorithm ends here for K=2, and finally, we obtained the result having of two clusters {1,2} and {3,4,5,6,7}, which remain unchanged (Fig. 3.1).

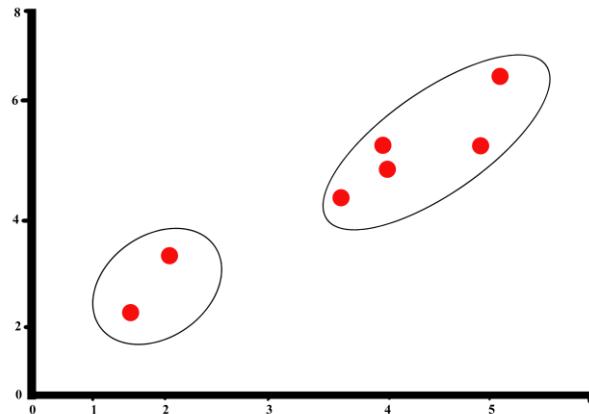


Fig. (3.1). Finally selected data points in two clusters after k-means algorithm (k=2).

Table 3.5. Dataset after step 4.

Individual	Centroid1	Centroid 2
1	0.56	5.02

(Table 3.5) cont.....

2	0.56	3.92
3	3.05	1.42
4	6.60	2.20
5	4.16	0.41
6	4.78	0.61
7	3.75	0.72

- **Example with Programming Approach**

Here, we are working on a simple hands-on example to understand how k-means performs. This example has been adapted from the TutorialPoint website¹. We will first generate a 2-D dataset comprising four different clusters and then apply the k-means algorithm.

- First, import the necessary packages:

```
import matplotlib.pyplot as plt
import seaborn as sns;
sns.set()
import numpy as np
from sklearn.cluster import KMeans
```

- Following code generates a 2-D data, containing four initial clusters:

```
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples = 400, centers = 4,
cluster_std = 0.60, random_state = 0)
```

- Next, to visualize the dataset:

```
plt.scatter(X[:, 0], X[:, 1], s = 20);
plt.show()
```

- Next, create an object of Kmeans class and provide the number of clusters, train the model and to predict:

¹https://www.tutorialspoint.com/machine_learning_with_python/machine_learning_with_python_clustering_algorithms_k_means.htm

```
kmeans = KMeans(n_clusters = 4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
```

- Now, to plot and visualize the cluster's centroid data points selected by k-means algorithm:

```
from sklearn.datasets.samples_generator import make_blobs
X, y_true = make_blobs(n_samples = 400, centers = 4,
cluster_std = 0.60, random_state = 0)
```

- Now again to visualize the dataset:

```
plt.scatter(X[:, 0], X[:, 1], c = y_kmeans, s = 20, cmap =
'summer')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c = 'blue', s =
100, alpha = 0.9);
plt.show()
```

❖ The basic steps are as follows:

1. Initialize k means with random values
2. For a given number of iterations:
 - 2.1. Iterate through items:
 - 2.1.1. Find the mean closest to the item
 - 2.1.2. Assign item to mean
 - 2.1.3. Update mean

- Read Data**

Read the data as an input text file ('data.txt'). In this data, each line represents each item, which has a numerical value, comma-separated. Get any CSV data of this type from any open source site.

- After reading the data, convert them into list. Each list is again a nested list having the item values for each of the features.

```
def ReadData(fileName):
    f = open(fileName, 'r');
```

```

lines = f.read().splitlines();
f.close();
items = [];
for i in range(1, len(lines)):
    line = lines[i].split(',');
    itemFeatures = [];
    for j in range(len(line)-1):

        # Convert feature value to float
        v = float(line[j]);

        # Add feature value to dict
        itemFeatures.append(v);
    items.append(itemFeatures);
shuffle(items);
return items;

```

- **Initialization of the Means**

Let's initialize values of each mean in the scope of the values of the features. So, now find 'min' and 'max' for each feature.

```

def FindColMinMax(items):
    n = len(items[0]);
    minima = [sys.maxint for i in range(n)];
    maxima = [-sys.maxint -1 for i in range(n)];

    for item in items:
        for f in range(len(item)):
            if (item[f] < minima[f]):
                minima[f] = item[f];

            if (item[f] > maxima[f]):
                maxima[f] = item[f];

    return minima, maxima;

```

- Here the *minima* and *maxima* are the lists elements having the 'min' and 'max' values. Then randomly initialize the mean feature values in the range of the 'min' and 'max' values.

```

def InitializeMeans(items, k, cMin, cMax):
    # Initialize means to random numbers between

```

```

# the min and max of each column/feature
f = len(items[0]); # number of features
means = [[0 for i in range(f)] for j in range(k)];

for mean in means:
    for i in range(len(mean)):

        # Set value to a random float
        # (adding +-1 to avoid a wide placement of a mean)
        mean[i] = uniform(cMin[i]+1, cMax[i]-1);
return means;

```

- **Euclidean Distance**

In the next step, use Euclidean's distance as a distance measure of similarity.

```

def EuclideanDistance(x, y):
    S = 0; # The sum of the squared differences of the elements
    for i in range(len(x)):
        S += math.pow(x[i]-y[i], 2)
    #The square root of the sum
    return math.sqrt(S)

```

- **Updating the Means**

During the update of a mean, compute the average value of each feature for each of the items within the cluster. To find the average values, find ‘m’:

$$m = (m * (n - 1) + x)/n$$

where ‘m’ = mean value; n = no. of items within the cluster; x = value of each feature.

```

def UpdateMean(n,mean,item):
    for i in range(len(mean)):
        m = mean[i];
        m = (m*(n-1)+item[i])/float(n);
        mean[i] = round(m, 3);

    return mean;

```

- **Classification or Categorization of the Items**

To classify or categorize each data point, let's find its similarity to each means for the given datapoint. Then categorize each of them to the closest one.

```
def Classify(means,item):
    # Classify item to the mean with minimum distance
    minimum = sys.maxint;
    index = -1;

    for i in range(len(means)):

        # Find distance from item to mean
        dis = EuclideanDistance(item, means[i]);

        if (dis < minimum):
            minimum = dis;
            index = i;

    return index;
```

- **Finding the Means**

Let's write a nested loop for finding means among all the data points, thereby categorizing each point to their closest cluster, updating the cluster's mean. Repeat this process until a predefined number of iterations or set any stopping criteria for termination of the program if the value is not updated after a specified number of iterations.

```
def CalculateMeans(k,items,maxIterations=100000):

    # Find the minima and maxima for columns
    cMin, cMax = FindColMinMax(items);

    # Initialize means at random points
    means = InitializeMeans(items,k,cMin,cMax);

    # Initialize clusters, the array to hold
    # the number of items in a class
    clusterSizes= [0 for i in range(len(means))];

    # An array to hold the cluster an item is in
    belongsTo = [0 for i in range(len(items))];
```

```

# Calculate means
for e in range(maxIterations):

    # If no change of cluster occurs, halt
    noChange = True;
    for i in range(len(items)):

        item = items[i];

        # Classify item into a cluster and update the
        # corresponding means.
        index = Classify(means,item);

        clusterSizes[index] += 1;
        cSize = clusterSizes[index];
        means[index] = UpdateMean(cSize,means[index],item);

        # Item changed cluster
        if(index != belongsTo[i]):
            noChange = False;

        belongsTo[i] = index;

    # Nothing changed, return
    if (noChange):
        break;

return means;

```

- **Finding the Clusters**

In the end, to find the clusters, let's iterate through all the data points, and thereby updating the space by categorizing each data to the closest cluster.

```

def FindClusters(means,items):
    clusters = [[] for i in range(len(means))]; # Init
clusters

    for item in items:

        # Classify item into a cluster
        index = Classify(means,item);

        # Add item to cluster
        clusters[index].append(item);

```

```
    return clusters;
```

The additional similarity measures are:

- **Cosine Distance:** Cosine distance measure reveals the cosine of the angle between the point vectors of the two points in the n-dimensional space.
- **Manhattan Distance:** Manhattan distance measure reveals the sum of the absolute differences between the two data points' coordinates.
- **Minkowski Distance:** Minkowski distance measure, also called as generalized distance metric'. This measure can be employed in both ordinal and quantitative variables.

3.2.4. Weakness of K-means Clustering

- 1) If the number of data is not that large, the initial grouping cluster decides to be large.
- 2) The number of clusters 'K' must be decided in advance. The downside is that each run won't give the same results, leaving the cluster to rely on initial random assignments.
- 3) We don't know of any real clusters that use the same data. This is because input in a different order can form different clusters if the data size is small.
- 4) It is sensitive to the early stages. Different starting positions can give different results to the cluster. Algorithms can be stuck on a local optimal.

3.2.5. Strength and Application of K-means Clustering

- 1) Relatively efficient and fast. Calculate the result for "O(tkn)", where n is the number of objects or points, 'k' is the cluster's number, and 't' is the iterations.
- 2) It can be applied to any application of machine learning or data mining.

- 3) Transforming the waveform into one of the K classes (called "vector quantization or image segmentation") is used for acoustic data in the speech identification.
- 4) It is also used in old-school graphic display devices and "image quantization" to select a color palette.

3.2.6. Programming Approach For K-means Clustering

```
import matplotlib.pyplot as plt
# from kneed import KneeLocator

# To create a synthetic dataset
from sklearn.datasets import make_blobs

# Kmeans Clustering algorithm import
from sklearn.cluster import KMeans

from sklearn.metrics import silhouette_score
from sklearn.preprocessing import StandardScaler

# Create synthetic data set
# make_blobs() returns a tuple of two values:
## A 2-D NumPy array with the x- and y-
values for each of the samples
#
#                                         A 1-D
# NumPy array containing the cluster labels for each sample

samples, labels = make_blobs(
    n_samples=100,
    centers=3,
    cluster_std=2.75,
    random_state=0
)

# Visualize samples and labels
print(samples[:6])
print(labels[:6])

[[ 3.64089227  0.32484208]
 [-1.48626296  5.3677692 ]]
```

```
[ 4.66021043  0.4713859 ]
[-3.461949   3.71875156]
[-1.90725058  0.398738   ]
[ 0.01951167  4.73374699] ]
[1 0 1 0 0 0]

# Standardize values for better clustering
scaling = StandardScaler()
scaled_samples = scaling.fit_transform(samples)

# Initiate the K-means-Cluster Algorithm
kmeans_clustering = KMeans(
    init="random",
    n_clusters=3,
    n_init=10,
    max_iter=200,
    random_state=0
)

# Feed the data samples to the algorithm
kmeans_clustering.fit(scaled_samples)

KMEANS (ALGORITHM='AUTO',           COPY_X=True,          INIT='RANDOM',
MAX_ITER=200,          N_CLUSTERS=3,          N_INIT=10,          N_JOBS=None,
PRECOMPUTE_DISTANCES='AUTO',      RANDOM_STATE=0,        TOL=0.0001,
VERBOSE=0)

# Evaluate the Alogrithm

# Lowest possible SSE (Sum of Squared Error)
kmeans_clustering.inertia_

84.89302060953563
# The determined final centers of clusters
kmeans_clustering.cluster_centers_
array([[ 0.7085658 ,  0.57518068],
       [ 0.21810856, -1.15192112],
       [-1.01524508,  0.50484285]])

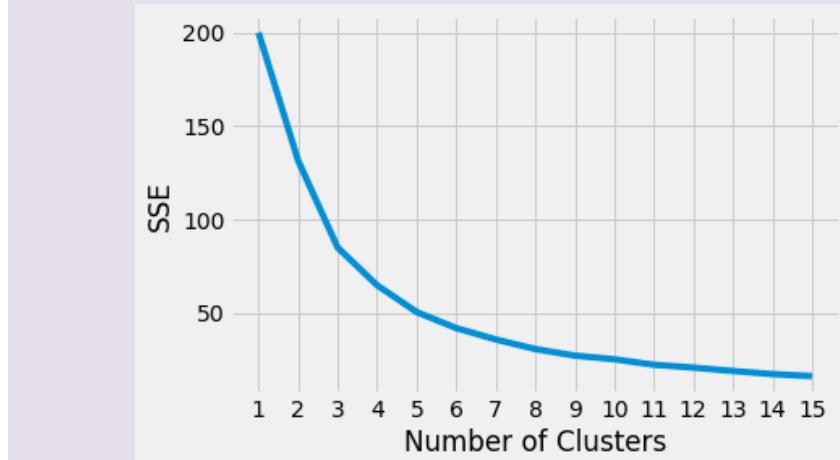
# Show number of iterations to converge
kmeans_clustering.n_iter_
```

```
11
# Optimize cluster algorithm by finding the best number of c
lusters

kmeans_kwargs = {
    "init": "random",
    "n_init": 15,
    "max_iter": 200,
    "random_state": 0,
}

# A list of SSE Values for each number of cluster
sse_list = []
for k in range(1, 16):
    kmeans_clustering = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans_clustering.fit(scaled_samples)
    sse_list.append(kmeans_clustering.inertia_)

plt.style.use("fivethirtyeight")
plt.plot(range(1, 16), sse_list)
plt.xticks(range(1, 16))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```



3.3. HIERARCHICAL CLUSTERING

3.3.1. Overview

The unlabeled data points in the unsupervised machine learning algorithm are clustered by the Hierarchical Clustering method. This clustering method has a predetermined ordering from top to bottom.

How to Perform?

1. Initially, each data point is treated as a cluster.
2. We initially denote the number of clusters as K.
3. Create a cluster by combining two adjacent data points as a result of K-1 clustering.
4. Combine two adjacent clusters resulting from K-2 clustering to create more clusters.
5. Repeat step 4 above until one large cluster is formed.
6. Dendograms are used to divide into clusters as they form.

❖ Types of Hierarchical Clustering

1. Divisive Clustering

The divisive clustering, also called as the top-down clustering method, applies all observations to a single cluster and then divides the cluster into at least two identical clusters.

2. Agglomerative Clustering

Agglomerative clustering is also known as the bottom-up clustering method. Each observation is assigned to its cluster.

The main differences between K means and Hierarchical Clustering are mentioned in Table 3.6.

Table 3.6. Difference between K-means and hierarchical clustering.

K-mean Clustering	Hierarchical Clustering
K-means clustering uses the distance between the data point and centroid to form different clusters. The clusters are spherical shapes.	Hierarchical clustering can be either division or agglomerative.
The value of K should be known.	Depending on the dendrogram, one can stop at any number of clusters.
Advantages <ul style="list-style-type: none"> • Guaranteed convergence. • Can cluster different sizes and shapes. 	Advantages <ul style="list-style-type: none"> • Can handle any similarity in distances. • Applicable to any attribute type.
Disadvantages Prediction of K-Value is difficult.	Disadvantage With large data sets, it could be expensive and slow.

3.3.2. ALGORITHMIC FRAMEWORK

1. Single Linkage

When all the data points are added one by one in a particular cluster, single linkage clustering is used. The distance between the two clusters is defined as the distance between their two nearest data points.

The formula to find the same is:

$$L(a, b) = \min(D(x_{ai}, x_{bj}))$$

2. Complete Linkage

When the clusters are clearly segregated and very compact, then complete linkage is used. The inter-cluster distance may be considered as the longest distance within any two data points between the two clusters.

The formula for the same is:

$$L(a, b) = \max(D(x_{ai}, x_{bj}))$$

3. Simple Average

Simple average algorithms define the distance between the clusters as the average distance between each data point so that the weights of the two clusters are the same according to their final output.

$$L(a, b) = T_{ab} / (N_a * N_b)$$

T_{ab} : Total of all pairwise distances among the two clusters.

N_a and N_b : The clusters' size.

It creates a set of nested clusters, arranged as a hierarchical tree, pictured as a dendrogram. A tree-like diagram that exploits the sequences of merges or splits.

❖ Dendrogram

When we visualize the connection or relationships among all the clusters, we draw a tree-like structure known as a dendrogram. Fig. (3.2) shows the tree-like hierarchical clusters and Fig. (3.3) shows the concentric clusters. With the increase in the length of lines of two clusters, the two clusters get farther. We concentrate on the distance between two objects to correctly interpret a dendrogram [7].

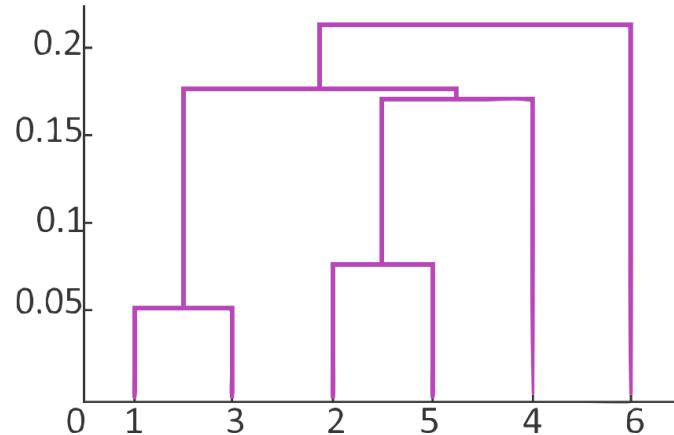


Fig. (3.2). Dendograms.

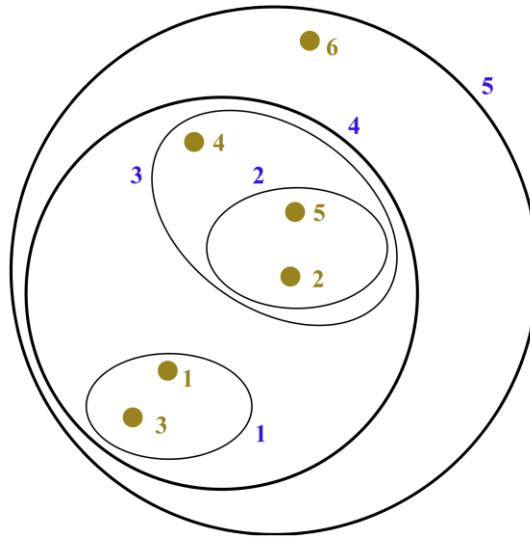


Fig. (3.3). Concentric clusters.

❖ Parts of a Dendrogram

It is essential to know all the parts of a dendrogram.

- Clades: The sections of the dendrogram are called the Clades. They are organized in the way of similarity.
- Each of the clades may have one or more leaves.

❖ Advantages of Hierarchical Clustering

1. It doesn't assume the number of clusters.
2. The desired cluster can be obtained by 'cutting' the dendrogram at the correct level.
3. Hierarchical clustering can be related to semantic classification.
4. Examples in biology (*e.g.*, phylogenetic reconstructions, *etc.*), web (*e.g.*, product catalogs, *etc.*).

❖ The Complexity of Hierarchical Clustering

1. The distance matrix is used to decide which cluster to merge or split.
2. Least-squares of the number of data points.
3. It cannot be used for large data sets.

3.3.3. Hands-on Example

3.3.3.1. Agglomerative Clustering Algorithm

Agglomerative clustering is the most popular hierarchical clustering technique. The algorithmic framework is as follows:

1. Calculate distance matrix between input data points.
2. Clustering each data point.
3. Repeat the following:
 - i) Merge two adjacent (closest) clusters
 - ii) Update distance matrix
4. Run until there is no single cluster left.

The main task is to calculate the distance between two clusters. Different definitions of distance measures between clusters result in different algorithms. Let us see a hands-on example to perform agglomerative clustering.

1. Input Setting

First, let us start with clusters of single data points and a distance/proximity matrix. Let us take some data points p₁, p₂, p₃, and so on. The circles represent the data points. The chart is the proximity matrix, as seen in Fig. (3.4).

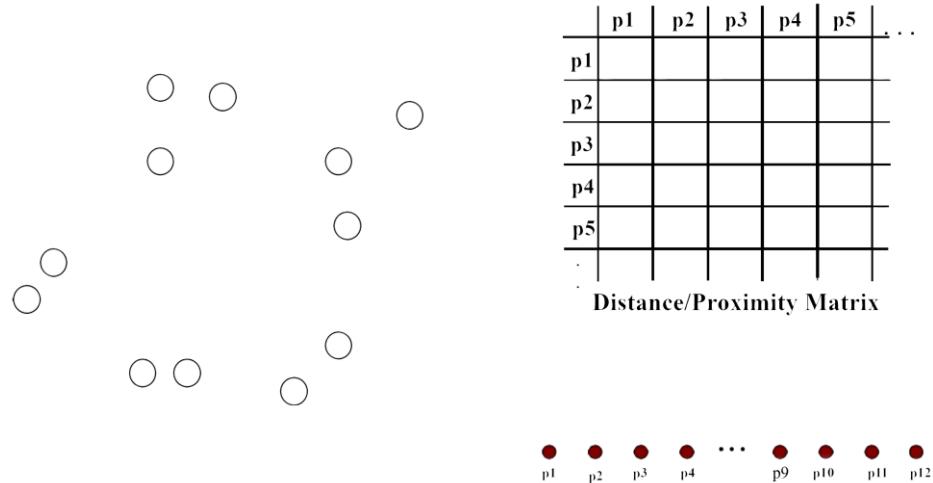


Fig. (3.4). Initial data points, blank clusters and initial proximity matrix.

2. Intermediate State

After the initial steps of merging, we get few clusters. Each cluster is named C1, C2, C3, C4, and C5. Now the matrix is changed to the cluster's proximity matrix. The datapoints p1, p2, ..., p12 are now linked to a particular cluster (Fig. 3.5).

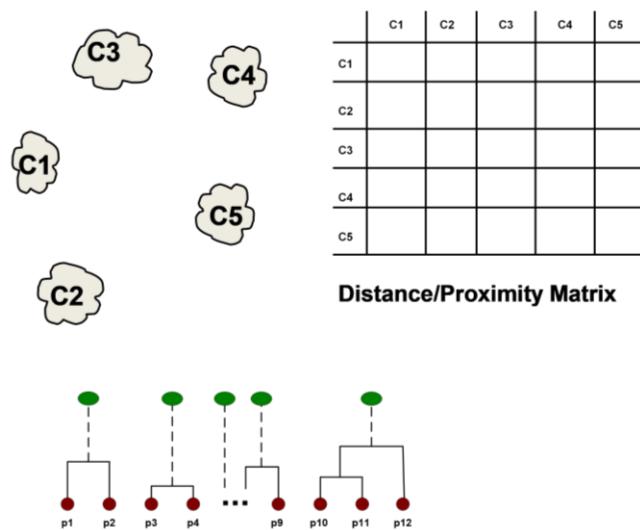


Fig. (3.5). Intial assignment of clusters and data points.

3. Intermediate State

Let us merge any of the two closest clusters in the next step, for example, here, C2 and C5. Now, we need to update the distance matrix in a proper cluster manner. Now the points are further linked to another level of clustering (Fig. 3.6).

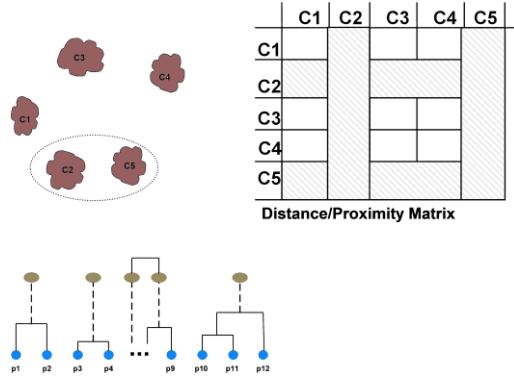


Fig. (3.6). After step 3.

4. After Merging State

After incremental updating and merging, we will obtain the merged clusters and the points linked to the root cluster (Fig. 3.7).

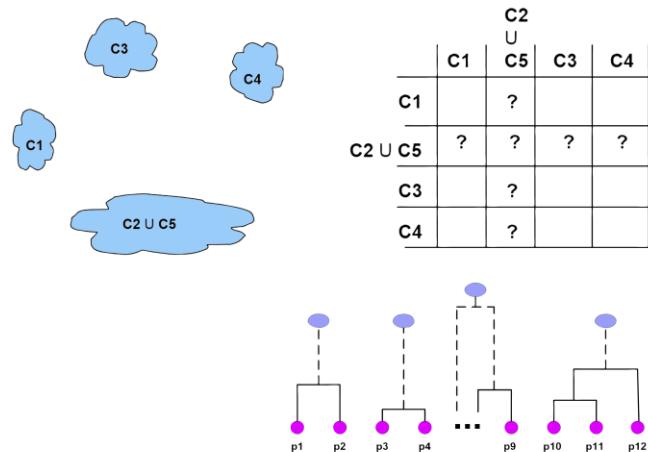


Fig. (3.7). After merging step.

❖ Distance Between Two Clusters

So ultimately, each cluster is a set of points. To find the distance between two clusters, there are several alternative approaches. Here, we will see three approaches, single-link, complete link, and group-average clustering.

▪ **Single-link Distance:** The distance between C_i and C_j clusters is the minimal distance between any data point in C_i and any data point in C_j . The distance can be framed by the two most similar data points, as follows:

$$D_{sl}(C_i, C_j) = \min_{x,y} \{d(x, y) | x \in C_i, y \in C_j\}$$

The nested cluster structure and dendrograms for the Single-link distance metric are given below (Figs. 3.8 and 3.9).

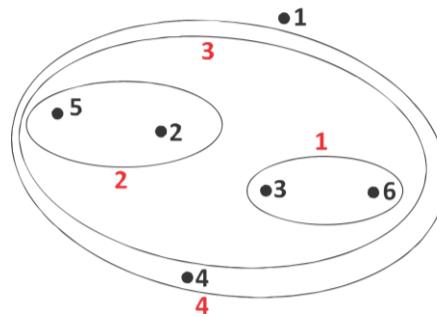


Fig. (3.8). Nested cluster showing the final set of data points.

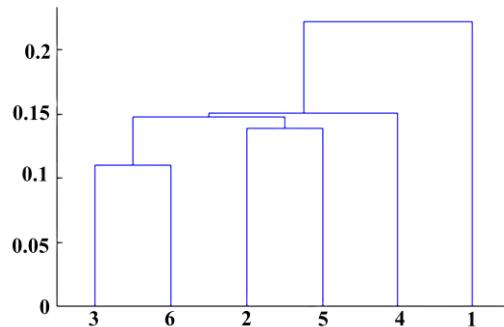


Fig. (3.9). Dendrograms for Single-link measure.

As per its strength is concerned, it can handle non-elliptical shapes, whereas it is sensitive to noise and outliers and produces long, elongated clusters.

▪ **Complete-link Distance:** The distance between C_i and C_j clusters is the maximal distance between any data point in C_i and any data point in C_j . The distance measure can be framed by the two most different data points, as follows (Figs. 3.10 and 3.11):

$$D_{sl}(C_i, C_j) = \max_{x,y} \{d(x, y) | x \in C_i, y \in C_j\}$$

The nested cluster structure and dendrograms for the complete-link distance metric are given below.

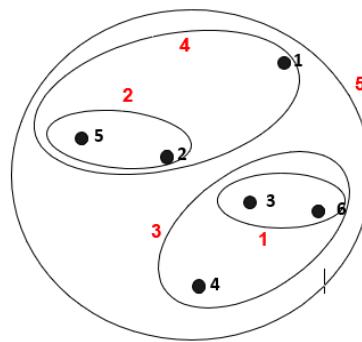


Fig. (3.10). Nested cluster for complete link.

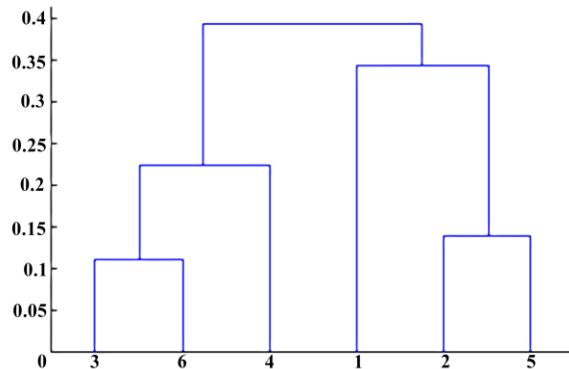


Fig. (3.11). Dendrograms for Complete-link measure.

As per its strength is concerned, it is more balanced clusters (with equal diameter) and less susceptible to noise, whereas it tends to break large clusters and all clusters tend to have the same diameter, *i.e.*, small clusters are merged with larger ones.

▪ **Average-link Distance:** The distance between the clusters C_i and C_j is the average (mean value) of pairwise proximity (closeness) between the two clusters. The distance measure can be framed by the two most similar data points, as follows (Figs. 3.12 and 3.13).

$$D_{sl}(C_i, C_j) = \text{mean}_{x,y} \{d(x, y) | x \in C_i, y \in C_j\}$$

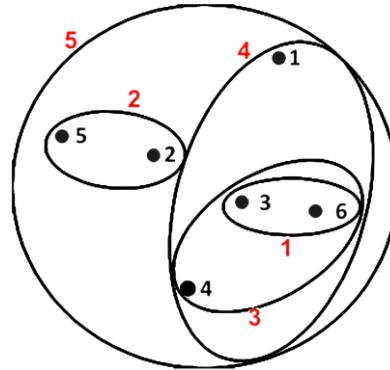


Fig. (3.12) Concentric nested cluster for average link measure.

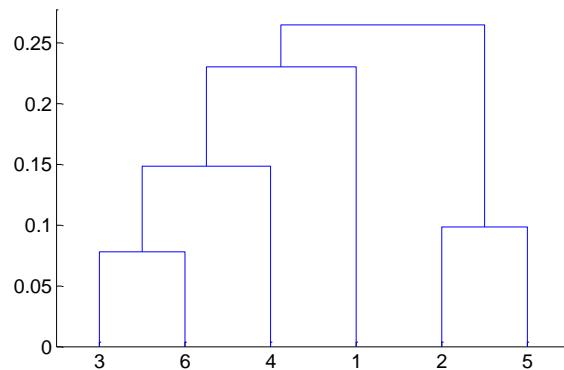


Fig. (3.13). Dendrograms for Average-link measure.

The Average-link distance is the settlement between single and complete link distance clustering methods. As par its strength is concerned, it is less susceptible to noise and outliers, whereas it is biased towards globular clusters.

3.3.4. Programming Approach for Hierarchical Clustering

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

# For creating a synthetic dataset
from sklearn.datasets import make_blobs

# For the cluster algorithm
from sklearn.cluster import AgglomerativeClustering

# For normalizing the data
from sklearn.preprocessing import StandardScaler, normalize

# For evaluating the clustering performance
from sklearn.metrics import silhouette_score

# Create synthetic dataset
samples, labels = make_blobs(n_samples=400, centers=3, n_features=2, random_state=0)
print(samples[:5])
print(labels[:5])

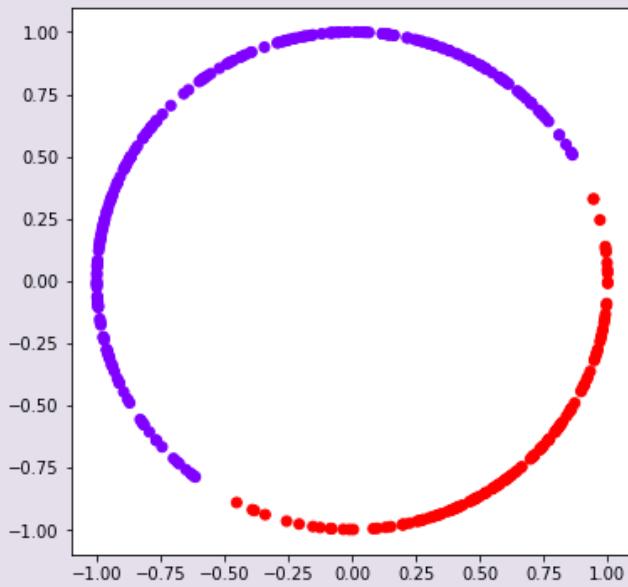
[[-2.08121364  2.4472446 ]
 [-1.89760802  1.97226647]
 [ 2.11567076  3.06896151]
 [ 1.40252881  4.98069536]
 [-0.21258918  3.79697097]]
[2 2 0 0 0]

# Make the data samples comparable by scaling
scaling = StandardScaler()
scaled_samples = scaling.fit_transform(samples)

# Normalizing the data for gaussian distribution
normalized_samples = normalize(scaled_samples)

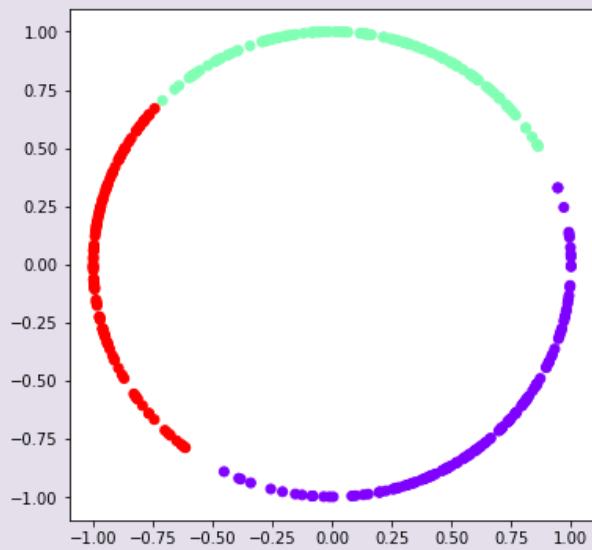
# Numpy array to pandas DataFrame for visualization
normalized_samples = pd.DataFrame(normalized_samples)
```

```
print(normalized_samples[:5])  
  
0      1  
0 -0.996463 -0.084027  
1 -0.956947 -0.290264  
2  0.969678  0.244386  
3  0.383107  0.923704  
4 -0.464855  0.885387  
  
# Optimize the clustering algorithm by setting different K's  
# for clustering  
  
# Initiate the cluster algorithm with n=2  
cluster_alg_1 = AgglomerativeClustering(n_clusters=2, affinity='euclidean', linkage='ward')  
  
plt.figure(figsize=(6, 6))  
plt.scatter(normalized_samples[0], normalized_samples[1],  
           c = cluster_alg_1.fit_predict(normalized_samples),  
           cmap ='rainbow')  
plt.show()
```



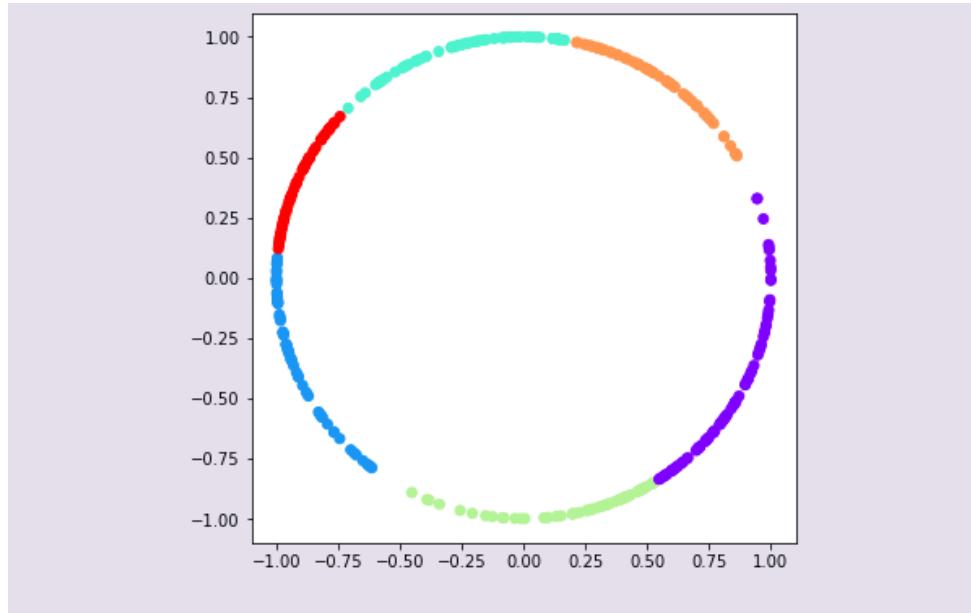
```
# Initiate the cluster algorithm with n = 3
cluster_alg_2 = AgglomerativeClustering(n_clusters=3, affinity='euclidean', linkage='ward')

plt.figure(figsize =(6, 6))
plt.scatter(normalized_samples[0], normalized_samples[1],
            c = cluster_alg_2.fit_predict(normalized_samples),
            cmap ='rainbow')
plt.show()
```



```
# Initiate the cluster algorithm with n = 6
cluster_alg_3 = AgglomerativeClustering(n_clusters=6, affinity='euclidean', linkage='ward')

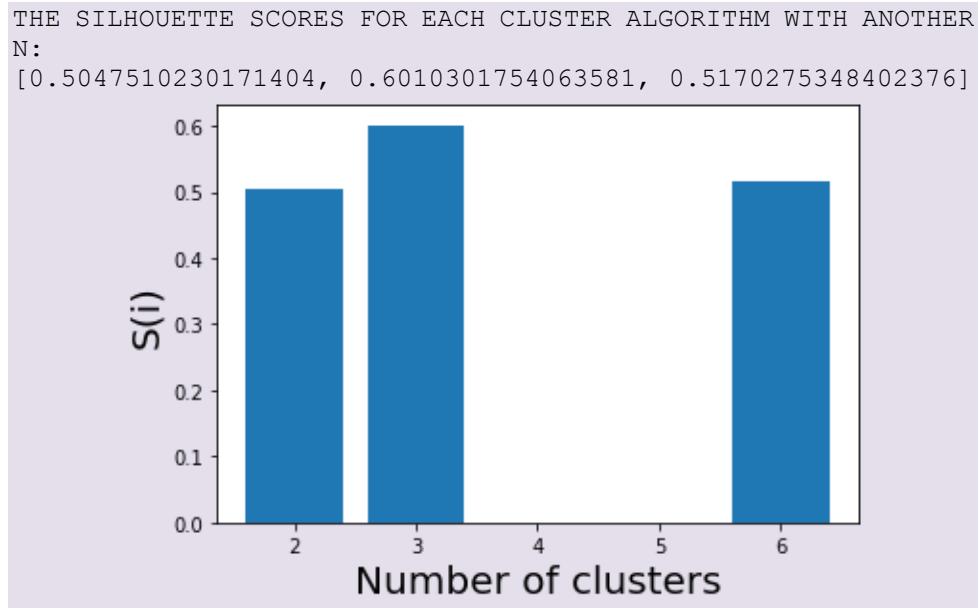
plt.figure(figsize =(6, 6))
plt.scatter(normalized_samples[0], normalized_samples[1],
            c = cluster_alg_3.fit_predict(normalized_samples),
            cmap ='rainbow')
plt.show()
```



```
k = [2, 3, 6]

# Calculate and add the silhouette scores of the different models to a list
silh_scores = []
silh_scores.append(
    silhouette_score(normalized_samples, cluster_alg_1.fit_predict(normalized_samples)))
silh_scores.append(
    silhouette_score(normalized_samples, cluster_alg_2.fit_predict(normalized_samples)))
silh_scores.append(
    silhouette_score(normalized_samples, cluster_alg_3.fit_predict(normalized_samples)))

print("The Silhouette Scores for each cluster algorithm with another n: ")
print(silh_scores)
# Plotting a bar graph to evaluate the results
plt.bar(k, silh_scores)
plt.xlabel('Number of clusters', fontsize = 20)
plt.ylabel('S(i)', fontsize = 20)
plt.show()
```



3.4. SELF-ORGANIZING MAP

3.4.1. Overview

Back in the 1970s, the self-organizing map, also known as Kohonen Map, was inspired by a biological neural system. It is an artificial neural network that follows an unsupervised learning approach. Its training is done through a competitive learning algorithm. SOM is one of the best clustering methods for the data spread in a multidimensional map. It also performs mapping on the data points. It reduces all complex problems into very simple and easy to interpret situations. SOM has two layers, the input layer and the output layer. The architecture of the self-organizing (SOM) map having two clusters and 'n' number of inputs may be seen in Fig. (3.14).

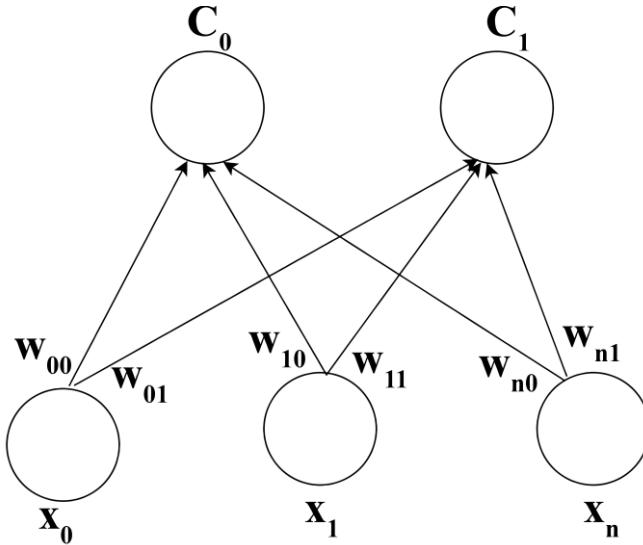


Fig. (3.14). The nodes in the SOM structure.

3.4.1.1. How SOM Works?

Suppose input data has a size of mn with m number of training instances and n number of features or variables.

At first, set the initial weights having size (n, C) , where ' C ' is the number of clusters. After initialization, iterate through each training instance, updating the winning vector, *i.e.*, the updating weight vector having the shortest distance from other training instances. The formula of weight update is:

$$w_{ij} = w_{ij}(old) - \alpha(t) * (x_{ik} - w_{ij}(old))$$

Here, alpha (α) is known as the learning rate at time ' t '; ' j ' represents the winning vector; ' i ' represents the i th feature of each instance; ' k ' represents the k th instances. After completing the training part of the self-organizing map, the new weights that are trained are now used for clustering new data samples. When a new data sample comes, it chooses the cluster having a winning vector (Fig. 3.15).

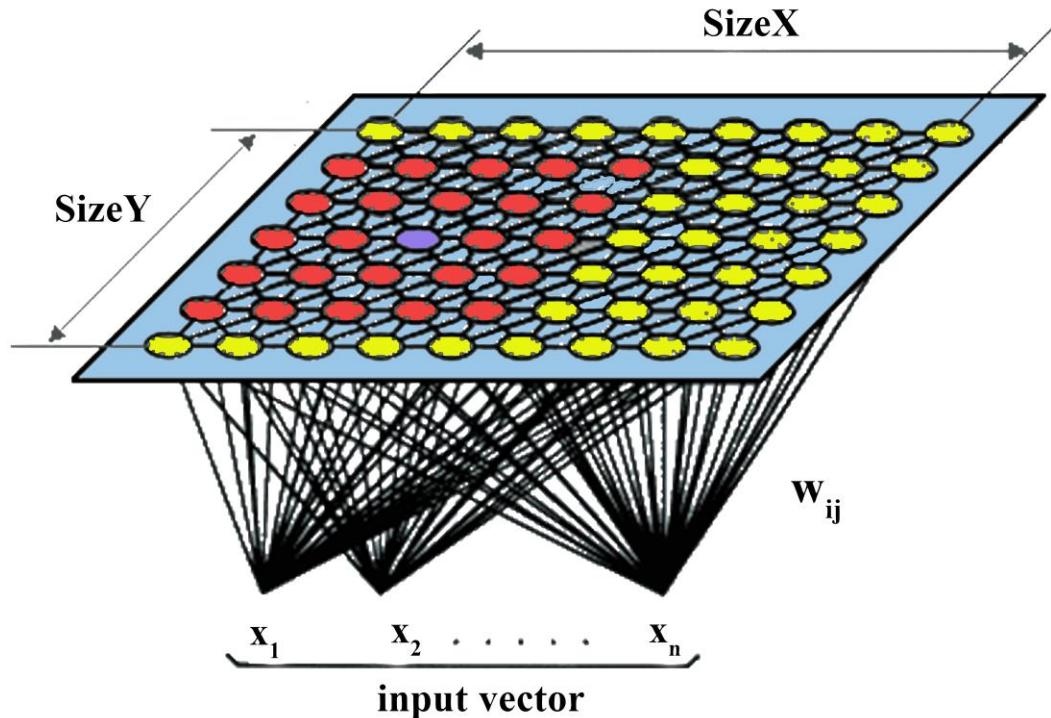


Fig. (3.15). The Self-organizing map projecting the data points on the data plane.

3.4.2. Algorithmic Framework

The basic algorithm is as follows. The steps involved are:

1. Set the initial weights
2. For iteration from 1 to N:
 - a) Select each training instance
 - b) Calculate the winning vector
 - c) Keep updating the winning vector
3. Repeat steps I, II, III for each of the training instances
4. Finally, cluster the new instances (test set)

3.4.3. Advantage and Disadvantage of SOM

3.4.3.1. Advantage

1. The nice thing about SOMs is that they are simple to comprehend. It is as easy as that: if they are near together and there is grey area between them, they are comparable. They are distinct if there is a dark ravine between them. Data can rapidly learn how to utilize self-organizing maps in contrast to Multidimensional Scaling and N-land.
2. As I have demonstrated, they efficiently categorize data and then assess their quality, allowing you to compute how excellent a map is and how strong the similarities between items are.

3.4.3.2. Disadvantage

1. Getting the correct data is a big issue with SOMs. Unfortunately, to construct a map, you'll require a value for each dimension of each sample member. This is a limiting characteristic of the usage of SOMs, commonly referred to as missing data, because it is not always possible and frequently quite challenging to obtain all of this data.
2. SOMs arrange sample data so that comparable samples are generally surrounded by similar samples in the result, although similar samples are not necessarily close to each other. If you have many purple hues, you won't always get one giant cluster with all the purples in it; occasionally, the clusters will divide, and you'll get two purple groupings.
3. They are computationally costly, which is a significant disadvantage since the dimensions of data become large, and dimension reduction visualization techniques become more relevant. However, the time required to calculate them grows as well. The more neighbors you employ to compute the distance for that black and white similarity map, the better the similarity map you'll receive, but the amount is limited.
4. The more neighbors you utilize to determine the distance for that black and white similarity map, the better the similarity map you'll receive, but the number of distances the method needs to compute grows exponentially [7].

3.4.3.3. Different Perspective of SOM

Among various perspectives, these four perspectives are popular for self-organizing maps:

1. The SOM is a representation of a new paradigm in artificial intelligence and cognitive modeling:

a) The SOM may be considered as an adaptive knowledge representation system and a model of unsupervised learning. The SOM is a clustering approach that maps related data samples to neighboring neurons and a projection approach that projects high-dimensional data space into low-dimensional space.

2. The SOM is a model of particular characteristics of biological neural networks:

a) The SOM is most commonly thought of as an artificial neural network model of the brain, particularly of empirically discovered ordered mappings in the cortex. The SOM hypothesis captures some of the brain's core processing principles backed up by much neurophysiological research.

3. The self-organizing maps are one of the best tools for statistical analysis and data visualization:

a) The mathematics and statistical methods of SOM are methodologically and computationally sound.

4. The SOM is a tool for sophisticated application development.

For complicated data sets, the SOM is commonly utilized as a data mining and visualization approach. In business and medical, applications include image processing and speech recognition, process management, economic analysis, and diagnostics.

3.4.4. Programming Approach for K-nearest Neighbor

```
import math
class SOM :
    # This is the function for computing the winning vector
    using Euclidean distance
```

```

def winning(self, w, data):
    D0 = 0
    D1 = 0

    for i in range(len(data)):
        D0 = D0 + math.pow((data[i] - w[0][i]), 2)
        D1 = D1 + math.pow((data[i] - w[1][i]), 2)

    if D0 > D1 :
        return 0
    else :
        return 1

# Function here updates the winning vector
def updation(self, w, data, J, alpha):

    for i in range(len(w)):
        w[J][i] = w[J][i] + alpha * (data[i] - w[J][i])

    return w

# Main code for Driver
def main():

    # Training data samples ( m, n )
    T = [[1, 1, 0, 0], [0, 0, 0, 1], [1, 0, 0, 0], [0, 0, 1, 1]
    ]]

    m, n = len(T), len(T[0])

    # weight initialization ( n, C )
    w = [[0.2, 0.6, 0.5, 0.9], [0.8, 0.4, 0.7, 0.3]]

    # training
    ob = SOM()

    epochs = 3
    alpha = 0.5

    for i in range(epochs):
        for j in range(m):

            # training sample
            data = T[j]

            # Compute winner vector

```

```

J = ob.winning(w, data)

# Update winning vector
w = ob.updatation(w, data, J, alpha)

# classify test sample
s = [0, 0, 0, 1]
J = ob.winning(w, s)

print("The final test data appoints to cluster : ", J)
print("The final trained weights : ", w)

if __name__ == "__main__":
    main()

The final test data appoints to Cluster: 0
The final trained weights: [[0.6000000000000001, 0.8, 0.5,
0.9], [0.3333984375, 0.0666015625, 0.7, 0.3]]

```

As we have seen from the above example of SOM in the Pythonic approach. It is to be noted that we have used the Euclidian distance as a measure of it. However, we can change the distance measure for change in result, as described above in the overview section. Now, let us see a Python pseudo-code for SOM.

```

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

# Dataset
from sklearn.datasets import load_iris

# For normalizing the data
from sklearn.preprocessing import StandardScaler, normalize

# Functions for the Self Organizing Map

# Load example dataset
def load_data():
    iris = load_iris()
    X = iris.data
    y = iris.target

    # Split the dataset into training and test

```

```
    Data_train, data_test, lab_train,
lab_test = train_test_split(X, y, random_state=0)

    return Data_train, data_test, lab_train, lab_test


def node_closest(data, t, map, m_rows, m_cols):
#


def euclidian_distance(v1, v2):
#


def dist_manance(r1, c1, r2, c2):
#


def common_most(lst):
#


np.random.seed(1)

dimensions = 4
rows = 30
columns = 30
range = rows + columns
learn = 0.5
steps = 5000

# 1. load data
Data_train, data_test, lab_train, lab_test = load_data()

# 2. construct the SOM
print("Constructing an SOM having 30x30 nodes from the iris data")
create_map = np.random.random_sample(size=(rows,columns,dimensions))

for s in range(steps):
    if s % (steps/10) == 0: print("stepsize = ", str(s))
    pct_left = 1.0 - ((s * 1.0) / steps)
    current_range = (int)(pct_left * range)
    current_rate = pct_left * learn

    t = np.random.randint(len(Data_train))
    (bmu_row, bmu_col) = closest_node(Data_train, t, map, rows, columns)
```

```

for i in range(rows):
    for j in range(columns):
        if dist_man(bmu_row, bmu_col, i, j) < current_range:
            map[i][j] = map[i][j] + current_rate *
(Data_train[t] - map[i][j])

## Rename from here

# 3. build and display U-Matrix
print("Construct the U-Matrix from the SOM")
u_matrix = np.zeros(shape=(row,columns), dtype=np.float64)
for i in range(rows):
    for j in range(columns):
        v = map[i][j] # a vector
        sumDig = 0.0; ct = 0

        if i-1 >= 0:
            sumDig += dist_euclidean(v, map[i-1][j]); ct += 1
        if i+1 <= rows-1:
            sumDig += dist_euclidean(v, map[i+1][j]); ct += 1
        if j-1 >= 0:
            sumDig += dist_euclidean(v, map[i][j-1]); ct += 1
        if j+1 <= Cols-1:
            sumDig += dist_euclidean(v, map[i][j+1]); ct += 1

        u_matrix[i][j] = sumDig / ct
plt.imshow(u_matrix, cmap='red'
plt.show()

# 4. build and display reduced data
print("Allocate each data point to each node of the map ")
map1 = np.empty(shape=(rows,columns), dtype=object)
for i in range(rows):
    for j in range(columns):
        map1[i][j] = [] # empty list

for t in range(len(Data_train)):
    (m_row, m_col) = closest_node(Data_train, t, map, row, columns)
    map1[m_row][m_col].append(data_y[t])

label_map = np.zeros(shape=rRows, columns), dtype=np.int)
for i in range(rows):
    for j in range(columns):
        label_map[i][j] = most_common(map1[i][j], 3)

```

```
plt.imshow(label_map, cmap=plt.cm.get_cmap('terrain_r', 4))
plt.colorbar()
plt.show()
```

CONCLUDING REMARKS

This chapter introduced the readers to clustering algorithms as a part of unsupervised machine learning algorithms. It also described the state-of-the-art clustering algorithms. The elaborative definition of k-mean clustering, hierarchical clustering and the self-organizing map was presented. This chapter also defined the algorithmic framework of each algorithm with hands-on examples with detailed Python codes and outputs.

Regression: Prediction

Abstract: This chapter introduces the readers to the in-depth knowledge of regression analysis. Regression is a concept used both in statistics and computer science, specifically in machine learning. However, the concept remains unaltered, but the applications. This chapter will learn about two primarily used regression analysis algorithms, linear regression and logistic regression. Here, each of the algorithms will be described in detail, with hands-on application. We will also learn linear and logistic regression in a more elaborative way while demonstrating through Python program on a real-world dataset.

Keywords: Machine learning, Regression, Linear regression, Logistic regression, Regressor, Logit, Prediction, Time-series analysis.

4.1. INTRODUCTION TO REGRESSION

4.1.1. What Is Regression?

Regression analysis is a statistical method for studying and comprehending the connection between two or more variables of interest. The method used for regression analysis aids in determining the variables which are significant and which may be ignored. It is a statistical technique used in finance, investing, and other fields to assess the strength and nature of a connection between one dependent variable (typically represented by Y) and a set of other variables (known as independent variables), as seen in Fig. (4.1).

We must grasp the following terms for regression analysis to be an effective method:

- Dependent Variable: It is the one that we are attempting to figure out or predict.
- Independent Variables: These are variables that impact the analysis or target variable and offer us information about the variables' relationships with the target variable.

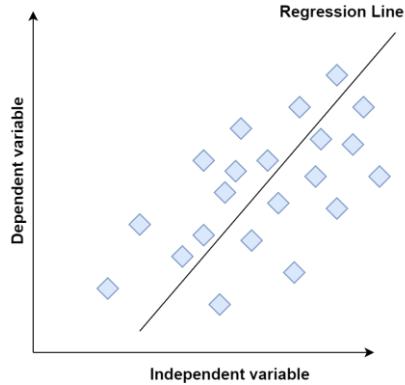


Fig. (4.1). The regression model.

The connection between a dependent variable and one or more independent variables is estimated using regression analysis. This approach is commonly used to predict outputs, forecast data, analyze time series, and determine causal effect relationships between variables. Based on the number of independent variables, the dimensionality of the regression line, and the kind of dependent variable, there are various types of regression procedures available. Linear regression and logistic regression are the two most used regression techniques. Some assumptions must be addressed for various forms of regression analysis and a grasp of the nature of variables and their distribution [8].

4.1.1.1. Linear Regression

Linear Regression is the most basic of all regression methods, attempting to establish connections between independent and dependent variables. The dependent variable is typically continuous in this case.

4.1.1.2. Logistic Regression

A supervised learning technique for classification, Logistic Regression, commonly known as Logit, Maximum-Entropy classifier, is a supervised learning method for classification. It uses regression to establish a relationship between dependent class variables and independent factors.

When it comes to regression analysis, what are the most common blunders? When dealing with regression analysis, it is critical to have a thorough understanding of the issue statement. We should presumably utilize linear regression if the issue description mentions forecasting. We should apply logistic regression if the issue description mentions binary categorization. Similarly, we must assess all of our regression models based on the issue statement.

4.1.2. How is it Different From Classification?

The distinction between classification and regression issues is significant. Fundamentally, classification and regression are both about predicting a label and a quantity.

Classification aims to predict a discrete class label. Regression, on the other hand, is the challenge of predicting a continuous variable. There is some overlap between the classification and regression techniques. A classification algorithm may predict a continuous value in the form of a probability for a class label. In contrast, a regression method may predict a discrete value in the form of an integer number.

Some algorithms, such as decision trees and artificial neural networks, may be utilized for classification and regression with minor tweaks. Some methods, such as linear regression for predictive regression modeling and logistic regression for classification predictive modeling, cannot readily be utilized for both issue types.

4.1.3. Applications of Regression

For prediction and forecasting, regression analysis is employed. This is closely related to the subject of machine learning. This statistical approach is utilized in a variety of sectors, including:

1. Stock market analysis: Understand the trend in stock prices, predict prices, and assess risks in the insurance sector in the financial industry.
2. Business: Understand the efficacy of marketing efforts, foresee pricing, and product sales.

3. Manufacturing: Evaluate the link between variables that affect the performance of a better engine.
4. Medicine: To create generic medications for illnesses, possible forecast of combinations of drugs.

4.2. LINEAR REGRESSION

4.2.1. Overview

Linear regression analysis is a statistical technique used for studying the relationship between variables. Linear regression studies the linear, additive relationships between all the variables.

Let us take an example. Consider ' Y_t ' as a "dependent" variable whose values we have to predict, and let us consider X_{1t}, \dots, X_{nt} as the "independent" variables. The equation can be framed as:

$$Y = B_0 + B_1X_1 + B_2X_2 + B_3X_3 + \dots + B_nX_n$$

According to this formula, the prediction for 'Y' will be a 'straight-line' function of each of the 'X' variables, keeping others constant, and the contributions of various X variables to the predictions are additive. The constants B_0, B_1, \dots, B_n The coefficients of the variables are the slopes of their unique 'straight-line' interactions with 'Y'. Other factors being fixed, B_i is the change in the expected value of 'Y' per unit of change in X_i . The so-called intercept, or extra constant B_0 , is the estimate that the model would make if all of the 'X' are zero. The coefficients and intercept are calculated using least squares, which means they are set to the distinctive values that reduce the sum of squared errors within the data sample to which the model is fitted [9]. The model's prediction errors are generally considered to be regularly distributed, independently, and identically.

The linear regression technique demonstrates a linear connection between a dependent 'Y' variable and one or more independent 'X' variables. Because linear

regression reveals a linear connection, it determines the change in the value of the dependent variable with changing the value of the independent variable.

As we know, linear regression is a supervised machine learning algorithm where the predicted output is continuous and has a constant slope. It is mainly used to predict values within a continuous range, such as stock price and commodity price, rather than classifying them into categories or classes, such as cars and airplanes. There are mainly two kinds of linear regression:

1. Simple linear regression
2. Multi-variable linear regression

4.2.1.1. Simple Regression

Simple linear regression is the most straightforward form of linear regression method. It uses the traditional method of slope-intercept to predict the most accurate prediction. The algorithm tries to learn this traditional method. There are two variables (let us say m and b), one input (*i.e.*, ' x ') in the slope-interception method. The output is the final prediction (let us consider ' y ') [9].

$$y = mx + b$$

EXAMPLE

Let's assume we have a dataset with the following columns (features): the amount of money a firm spends on television (TV) advertising each year and its yearly sales in units sold. Here, we are trying to frame an equation that will allow us to estimate how many units a firm will sell based on its spending on radio advertising. Companies are represented by the rows (observations) (Table 4.1).

Table 4.1. Sample data set for company sales report.

Company	TV (In Million USD)	Sales (In Million USD)
Company A	5.2	12.2

(Table 4.1) cont.....

Company B	3.1	13.6
Company C	4.6	17.7
Company D	7.1	11.4

4.2.1.2. Making a Prediction

Our prediction method generates a sales forecast with a company's radio advertising budget and current Weight and Bias settings.

4.2.1.2.1. Weight

Radio's independent variable's coefficient. We call coefficients as weights in machine learning.

4.2.1.2.2. Commercials on TV

It's the variable that's unrelated to the dependent one. These are referred to as features in machine learning.

4.2.1.2.3. Bias

The point where our line crosses the y-axis is known as the intercept. We can call intercepts as bias in machine learning. All of our forecasts are controlled by bias.

The right settings for Weight and Bias will be determined by our algorithm. Our equation will get close to the line of best fit after our training (Fig. 4.2).

```
def predict_sales(tv, weight, bias):
    return weight*tv + bias
```

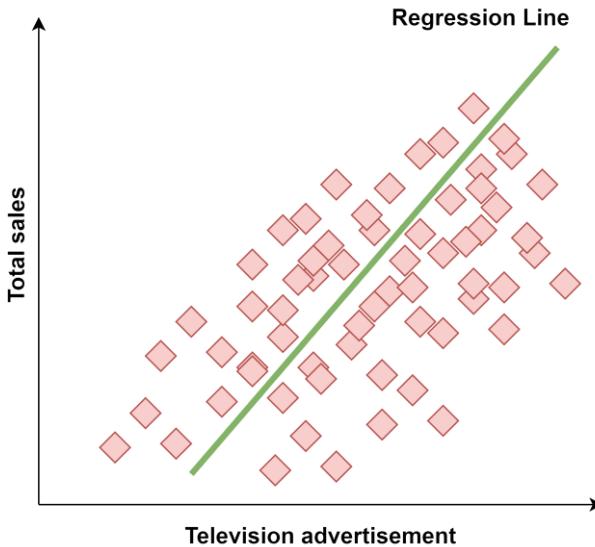


Fig. (4.2). Example showing trained regression model across TV advertisement and net sales.

4.2.1.3. Multi-variable Regression

Multi-variable regression is a bit complex regression method than the simple regression method. A multi-variable linear equation is easy to understand. Look at the equation given below:

$$f(x, y, z) = w_1x + w_2y + \dots + w_tz$$

In the given equation:

' w_i ' represents the coefficients or weight.

'x,y and z' represent the attributes or distinct pieces of information.

For example: if we choose sales predictions of Company A, Company B, and Company C, then the equation will be:

$$\text{Sales} = w_1 * \text{CompanyA} + w_2 * \text{CompanyB} + w_3 * \text{CompanyC}$$

EXAMPLE

Let's we assume, that few companies are giving advertisement on television, internet, and magazine. We aim to predict their sales in terms of products sold, observing the effect of advertisements on different media platforms (Table 4.2).

Table 4.2. Sample dataset showing selling of products based on advertisement.

Company	TV	Internet	Magazine	Products
Company A	100	25	49	12
Company B	130	31	27	29
Company C	90	25	14	17
Company D	170	55	29	27

4.2.1.3.1. Growing Complexity

When the number of features starts increasing in any model or example, the complexity of the model increases accordingly. Thus, if the complexity increases in our model, it would be challenging to visualize or comprehend the data.

To avoid the difficulty due to the increasing complexity, we can break the data into small parts and compare one or two features at a time.

4.2.1.3.2. Making Predictions

Our forecast function generates a sales prediction based on our current weights (coefficients) and a company's TV, radio, internet, and magazine advertisement spending. Our model will look for weight values that will reduce our cost function the most.

$$\text{Product Sales} = W_1 * \text{TV} + W_2 * \text{Internet} + W_3 * \text{Magazine}$$

```
def predict(features, weights):
    predictions=np.dot(features,weights)
    return predictions
```

4.2.2. Linear Regression Line

The linear regression line is a linear line that shows the relationship between a dependable and independent variable. This linear regression line can predict two types of relationships between the dependable and independent variables.

4.2.2.1. Positive Linear Relationship

In a positive linear relation, the value of the dependent variable rises on the Y-axis while the independent variable rises on the X-axis (Fig. 4.3).

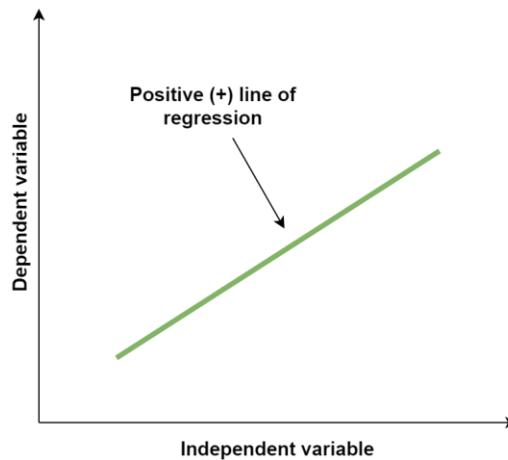


Fig. (4.3). Positive line of regression.

4.2.2.2. Negative Linear Relationship

A negative linear relationship occurs when the dependent variable falls on the Y-axis while the independent variable rises on the X-axis (Fig. 4.4).

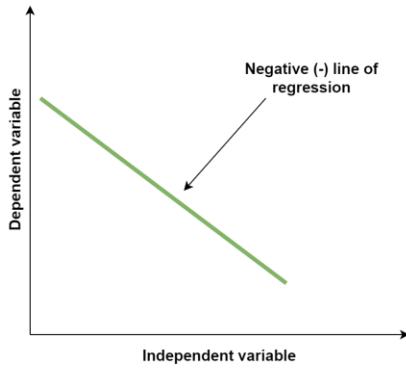


Fig. (4.4). Negative line of regression.

4.2.2.3. Assumptions in Regression and its Justification

As we know, every model or algorithm has several assumptions. Similarly, the regression method has some assumptions [10]. Let us understand them one by one:

1. As linear relationships are the most basic non-trivial relationships imaginable.
2. Since the “actual” relationships between our variables are frequently somewhat linear throughout the range of values.
3. Even though they’re not, we can frequently transform the variables to make the relationships linear. This is one of the strongest assumptions. Steps involved in the regression model are:
 - a. Look at the scatterplots of the variables first.
 - b. After the observation is complete, fit a model.
 - c. Analyze the errors and search for any non-linear patterns in the scatterplots once you’ve fitted the model.
 - d. If you see any non-linear trends, you can rule them out using variable transformations and derive a useful prediction using linear regression.

4.2.3. Regression Algorithms

There are five regression algorithms. Let us understand each one of them.

1. Ordinary least squares
2. Polynomial
3. Lasso
4. Ridge
5. Stepwise

1. Ordinary Least Squares

We use the Ordinary Least Squares (OLS) technique to do linear regression. The square of the difference from the mean is calculated for each data point in this technique, and the total loss function is minimized.

$$l = \sum_{i=1}^n (y_i - \hat{y})^2$$

2. Polynomial

The features are mapped in a polynomial form in polynomial regression. It's a revised form of linear regression in which the problem statement stays the same but the technique changes. The input vector has now been transformed to a higher-dimensional vector that acts as a type of pseudo-input vector.

$$x = (x_0, x_1) \xrightarrow{yields} x' = (x_0, x_0^2, x_1, x_1^2, x_0 x_1)$$

3. Lasso

Similar to vanilla regression, Lasso Regression attempts to decrease the conventional least-squares error by adding an additional term. Every data point's L1 norm is multiplied by a hyperparameter. This simplifies the model and avoids overfitting.

$$l = \sum_{i=1}^n (y_i - \tilde{y})^2 + \alpha \sum_{j=1}^p |w_j|$$

4. Ridge

Ridge regression lowers the model's complexity in the same way that lasso regression does. The sole difference is that the regularization term is now L2 rather than L1.

$$l = \sum_{i=1}^n (y_i - \tilde{y})^2 + \alpha \sum_{j=1}^p w_j^2$$

5. Stepwise

Stepwise regression (sometimes referred to as spline regression) is a technique for fitting a piecewise function to data. It's most commonly employed with linear models. It may also be generalized to a greater extent. The regression equation is written as H_α , where H_α is the shifted Heaviside step function with a discontinuity at α .

$$y = ax + b(x - \bar{x})H_\alpha + c$$

4.2.4. Programming Approach for Linear Regression

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

# Load Example Dataset and linear model from scikit-learn
```

```
from sklearn import datasets, linear_model

# For calculating metrics
from sklearn.metrics import mean_squared_error, r2_score

# Load diabetes dataset
X_data, y_data = datasets.load_diabetes(return_X_y=True)

# Use only one feature
X_data = X_data[:, np.newaxis, 2]

# Split into training and testing sets
X_train = X_data[:-20]
X_test = X_data[-20:]
y_train = y_data[:-20]
y_test = y_data[-20:]

# Create linear regression object
regressor = linear_model.LinearRegression()

# Train the model
regressor.fit(X_train, y_train)

# Make predictions
y_pred = regressor.predict(X_test)

# The coefficients
print('Coefficients: \n', regressor.coef_)

# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y_test, y_pred))

# 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_test, y_pred))

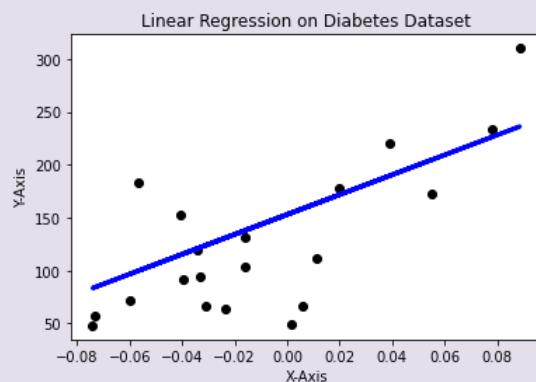
Coefficients:
[938.23786125]
Mean squared error: 2548.07
Coefficient of determination: 0.47

# Plot outputs
plt.scatter(X_test, y_test, color='black')
```

```
plt.plot(X_test, y_pred, color='blue', linewidth=3)

plt.xlabel("X-Axis")
plt.ylabel("Y-Axis")
plt.title("Linear Regression on Diabetes Dataset")

plt.show()
```



4.3. LOGISTIC REGRESSION

4.3.1. Overview

When it comes to machine learning algorithms, Logistic Regression is the most common. Because of its simple technique is utilized in binary classification and may be used on various platforms for a range of problem statements.

The logistic function, also known as the sigmoid function, was created by statisticians to characterize the features of population increase in ecology, such as how it rises fast and eventually reaches the environment's carrying capacity. It's an S-shaped curve that can translate any real-valued integer to a value between 0 and 1 but never precisely between those two points.

Where e is the natural logarithms' base and x is the numerical value to be transformed. The values between -5 and 5 have been converted into the range 0 and 1 using the logistic function, as shown below.

$$\frac{1}{1 + e^{-z}}$$

Binary classification techniques employ logistic regression. All of our observations assist us in allocating a distinct set of classifications. Unlike linear regression, which produces continuous numerical values, logistic regression produces a probability value translated to two or more discrete classes using the logistic sigmoid function.

Logistic regression, like linear regression, employs an equation as its representation.

To forecast an output value, input values (x) are blended linearly using weights or coefficient values (referred to as the Greek capital letter Beta (β)). The output (y) value being modeled is a binary value (0 or 1) rather than a numeric number, which is a significant distinction from linear regression [11].

An example of a logistic regression equation is shown below:

$$y = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

Where y is the anticipated output, β_0 is the bias or intercept term, and β_1 is the single input value coefficient (x). The β coefficient for each column in your input data must be learned from your training data.

4.3.1.1. Comparison to Linear Regression

Suppose you are given two different data sets and asked to perform linear and logistic regression. One data set is time spent studying, and the second is about exam scores of different students. Both the regression methods will predict different things:

Linear Regression: Predictions from the linear regression are continuous. Thus, they will help predict the score of students between the range of 0 to 100.

Logistic Regression: Predictions from the logistic regression are discrete; thus, it will help predict whether the student passed the exam or failed.

Types of logistic regression

- Binary type (Pass/Fail)
- Multi-type (Cats, Dogs, Sheep)

4.3.1.2. Binary Logistic Regression

When there are just two categories, binary classification is the simplest type of classification issue. Binary logistic regression aims to train a classifier that can determine a new input observation class using a binary choice.

Take a look at the example. The number of hours spent sleeping and studying determines whether a student passes or fails a test. Thus, the number of hours spent sleeping and studying are your characteristics, while passing (1) and failing (0) are the grades in such situations. The problem's data set is listed below Table 4.3.

Table 4.3. Table showing result of students based on the hours spent in study and sleep.

Study	Sleep	Test Result
5	8	1
4	3	0
8	5	1
7	6	1

4.3.1.3. Multiclass Logistic Regression

We did regression in the linear regression section, so we only had one output \mathbf{y} and attempted to make it as close to the real goal \mathbf{y} as feasible. Instead of regression, classification is used here, with each input \mathbf{X} being assigned to one of L classes.

So far, we've looked at the case where each training sample \mathbf{x} either belongs to a specific class ($\mathbf{y} = \mathbf{1}$) or doesn't ($\mathbf{y} = \mathbf{0}$). However, we can extend this concept to the scenario where \mathbf{x} can be a member of many classes at the same time, *i.e.*, $\mathbf{y} \in \{\mathbf{0}, \mathbf{1}\}^C$, where C is the number of classes. For example, if the inputs are images and we have three classes ("horse," "truck," and "plant"), we may say that a given image has a truck and a plant.

Here, we will expand our definition to $\mathbf{y} = \mathbf{0}, \mathbf{1}, \dots, \mathbf{n}$ and $\mathbf{y} = \mathbf{0}, \mathbf{1}, \dots, \mathbf{n}$ instead of $\mathbf{y} = \mathbf{0}, \mathbf{1}$ and $\mathbf{y} = \mathbf{0}, \mathbf{1}$. Essentially, we repeated the binary classification process multiple times, one for each class.

4.3.1.3.1. Procedure

1. Break the main problem into $(n + 1)$ binary problems (classification).
2. For each class:
 - i. Predict the probability that the observations are in that single class.
 - ii. Prediction = finding the maximum value of the probability of the categories or classes.

For each sub-problem, we select one class ($\mathbf{y} = \mathbf{1}$) and combine all the others into a second class ($\mathbf{y} = \mathbf{0}$). After that, we choose the class with the greatest estimated value.

4.3.1.4. Some Essential Functions and Concepts Used in Logistic Regression

Binary logistic regression with the Sigmoid function and multi-class logistic regression with a SoftMax are the most used methods.

1. Sigmoid Function

The sigmoid function (also known as the standard logistic function) is defined for a scalar real integer as:

$$\sigma^z = \frac{1}{1 + e^{-z}}$$

It returns values from the range of 0 and 1. Because of this characteristic, it may be used to interpret a real-valued score z as a probability.

2. Softmax Function

The softmax function is defined for a vector $x \in \mathbb{R}^n$ as:

$$\text{Softmax}(x_i) = \frac{e^{x_i}}{\sum_{j=1}^n e^{x_j}}$$

Each element in $\text{Softmax}(x_i)$ is compressed to fit within the range $[0, 1]$, and the total number of elements is 1. As a result, the softmax function helps convert a discrete probability distribution from an arbitrary vector of actual values.

3. Decision Boundary

A threshold can be established to forecast which class a data belongs to. The derived estimated probability is categorized into classes based on this threshold.

If the value which has been predicted is less than 0.5, categorize the email as spam; otherwise, label it as not spam.

There are two types of decision boundaries: linear and non-linear. To provide a complicated decision boundary, the polynomial order can be raised.

4. Cost Function

The cost function is different in the case of logistic regression compared to linear regression. The cost function used for linear regression is basically the "mean squared error (MSE)". If this is utilized for logistic regression, the function of parameters will be non-convex (theta). Only if the function is convex, then the gradient descent will lead to a global minimum.

$$\begin{aligned} \text{Cost function} &= J(\Theta) = J(h_\theta(x), y(\text{actual})) \\ &= -\log(h_\theta(x)) \quad [\text{if } y = 1] \\ &= -\log(1 - h_\theta(x)) \quad [\text{if } y = 0] \end{aligned}$$

4.3.2. Algorithmic Framework

The probability of the default class is modeled using logistics. Suppose we're modeling packages as cargo shipment or general shipment based on their weight, for example. In that case, the first class might be cargo, and the logistic regression model could be expressed as the probability of cargo given a package weight:

$$P(\text{package} = \text{cargo} | \text{weight})$$

To put it another way, we're modeling the probability that an input (X) belongs to the default class ($Y = 1$); we can write it down like this:

$$P(Y = 1 | X) = P(X)$$

In order to produce a probability prediction, the probability prediction must be converted into binary values (0 or 1). Although logistic regression is a linear approach, the logistic function is used to modify the predictions. As a result, we can no longer comprehend the predictions as a linear combination of the inputs, as we can with linear regression. Continuing from the previous example, the model may be expressed as:

$$y = \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}}$$

The above equation can be reversed as follows. By adding a natural logarithm (\ln) to the opposite side, we may eliminate the e from one side:

$$\ln\left(\frac{P(x)}{1 - P(x)}\right) = \beta_0 + \beta_1 x$$

This is important because we can see that the output on the right is calculated linearly again, and the input on the left is a log of the default class's probability.

The odds of the default class is the ratio on the left. Odds are computed as a ratio of the probability of an occurrence, divided by the probability of an event, which is not occurring. For example, $0.6/(1-0.6)$ has odds of 1.5. Instead, we could write:

$$\ln(Odds) = \beta_0 + \beta_1 x$$

We call the left-hand side the log-odds or the probit since the odds are log converted. Although various types of functions can be used for the transform, the link function, which connects the linear regression equation to the probabilities, is commonly referred to as the link function.

We may express it as if we shift the exponent back to the right.

$$Odds = e^{\beta_0 + \beta_1 x}$$

All of this demonstrates that the model is still a linear combination of inputs but that this linear combination is related to the default class's log-odds.

The logistic regression algorithm's coefficients (Beta values (β)) must be determined from your training data. The maximum-likelihood estimate is used for this.

Although it makes assumptions about the distribution of your data, maximum-likelihood estimation is a typical learning technique utilized by many machine learning algorithms.

The optimal coefficients would result in a model that predicted a value very close to 1 for the default class (*i.e.*, cargo package) and a value extremely close to 0 for the other class (*i.e.*, general package). The idea behind maximum-likelihood logistic regression is that a search method looks for coefficients that decrease the difference between the model's predicted probabilities and the values in the data.

4.3.2.1. Predict with Logistic Regression

It's as simple as putting numbers into the logistic regression equation and computing a result to make predictions with a logistic regression model.

Let's see an example. Assume we have a model that can predict whether a package will be transported by freight train or by general transportation depending on its weight. If the shipment weighs 700 kilograms, it will be transported by freight rail or ordinary passenger train.

The coefficients of $\beta_0 = -110$ and $\beta_1 = 0.5$ have been learned. We can compute the probability of a package with a weight of 250kg using the equation above ($P(\text{package_cargo}|\text{weight}=250)$).

$$\begin{aligned} y &= \frac{e^{\beta_0 + \beta_1 x}}{1 + e^{\beta_0 + \beta_1 x}} \\ &\approx y = \frac{e^{-110 + 0.5 \cdot 250}}{1 + e^{-110 + 0.5 \cdot 250}} \\ &\approx y = 0.999999694 \end{aligned}$$

Here, the probability is near 1, which means the package will be shipped *via* cargo rail.

4.3.2.2. Data Preparation for Logistic Regression

The assumptions that logistic regression makes about the distribution and connections in your data are pretty similar to those that linear regression makes. In the end, the goal of predictive modeling and machine learning initiatives is to make correct predictions rather than analyze the findings. As a result, as long as the model is resilient and performs well, you can break some assumptions.

1. Variable with a Binary Output: As we've already stated, logistic regression is designed for binary (two-class) classification issues. It will forecast the likelihood of an instance belonging to the default class, classified as 0 or 1.

2. Gaussian Distribution: Logistic regression is a linear method with a Gaussian distribution. It does, however, require a linear connection between the input and output variables. A more accurate model may be achieved using data transformations on your input variables to disclose this linear connection better.

3. Remove Outliers: Since logistic regression assumes no error in the output variable (y), consider eliminating outliers and maybe unclassified instances from your training data.

4. Remove Correlated Data Values: If you have several strongly correlated data values, the model can overfit, much like linear regression. Calculate the pairwise correlations between all inputs and exclude those that are highly correlated.

4.3.3. Programming Approach for Logistic Regression

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

# Load Example Dataset
from sklearn.datasets import load_iris

# For splitting into train and test dataset
from sklearn.model_selection import train_test_split

# Load Logistic Regression from Scikit-Learn
from sklearn.linear_model import LogisticRegression
```

```
# For calculating metrics
from sklearn.metrics import mean_squared_error, r2_score

# Get dataset
X_data, y_data = load_iris(return_X_y=True)

# Use only one feature
X_data = X_data[:, :2]

# Split into train and test data
X_train, x_test, Y_train, y_test = train_test_split(X_data, y_data, test_size=0.25, random_state=0)

# Initiate and Train Logistic Regressor
regression_classifier = LogisticRegression(random_state=0).fit(X_train, Y_train)

# Predict labels of classifier
predictions = regression_classifier.predict(x_test)
print("Predictions: " + str(predictions))

# Calculate probabilities of test data
probabilities = regression_classifier.predict_proba(x_test)
print("Probabilities: " + str(probabilities))

# Calculate the score
score = regression_classifier.score(x_test, y_test)
print("Score: " + str(score))

Predictions: [1 1 0 2 0 2 0 2 2 1 1 2 1 2 1 2 1 0 1 1 0 0 1 1 0 0 2
0 0 2 1 0 2 1 0 2 2 1 0
1]
Probabilities: [[8.78532904e-02 5.87258994e-01 3.24887716e-01]
[7.01767782e-03 6.65604088e-01 3.27378234e-01]
[9.57854812e-01 2.16742805e-02 2.04709071e-02]
[3.22859757e-04 1.38904034e-01 8.60773106e-01]
[9.11913061e-01 7.08308459e-02 1.72560932e-02]
...
...
...
[3.49142225e-01 5.41654473e-01 1.09203302e-01]
[7.58113591e-01 1.22837251e-01 1.19049158e-01]
[3.30693805e-02 5.67477098e-01 3.99453521e-01]]
Score: 0.7894736842105263
```

```

# The coefficients
print('Coefficients: \n', regression_classifier.coef_)

# The mean squared error
print('Mean squared error: %.2f'
      % mean_squared_error(y_test, predictions))

# 1 is perfect prediction
print('Coefficient of determination: %.2f'
      % r2_score(y_test, predictions))

Coefficients:
[[ -2.5238339   2.04057617]
 [  0.48047069 -1.37876396]
 [  2.04336321 -0.66181221]]
Mean squared error: 0.21
Coefficient of determination: 0.63

# Plot outputs

# Plot the decision boundary.
x_min, x_max = X_data[:, 0].min() - .5, X_data[:, 0].max() + .
5
y_min, y_max = X_data[:, 1].min() - .5, X_data[:, 1].max() + .
5
h = .02 # step size in the mesh
xx, yy = np.meshgrid(np.arange(x_min, x_max, h), np.arange(y_m
in, y_max, h))
Z = regression_classifier.predict(np.c_[xx.ravel(), yy.ravel()])
]

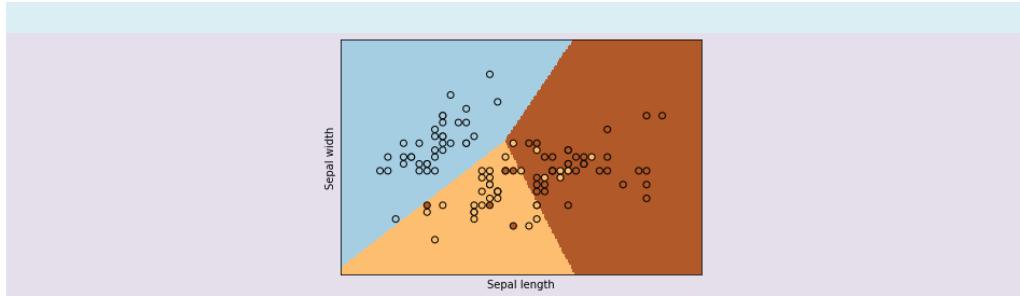
# Put the result into a color plot
Z = Z.reshape(xx.shape)
plt.figure(1, figsize=(6, 4))
plt.pcolormesh(xx, yy, Z, cmap=plt.cm.Paired)

# Plot also the training points
plt.scatter(X_train[:, 0], X_train[:, 1], c=Y_train, edgecolor
s='k', cmap=plt.cm.Paired)
plt.xlabel('Sepal length')
plt.ylabel('Sepal width')

plt.xlim(xx.min(), xx.max())
plt.ylim(yy.min(), yy.max())
plt.xticks(())
plt.yticks(())

plt.show()

```



CONCLUDING REMARKS

The readers were given a thorough understanding of regression analysis in the introductory chapter. The notion of regression is used in both statistics and computer science, particularly in machine learning. The principle, on the other hand, hasn't changed, although the applications have. The two most used regression analysis techniques, linear regression and logistic regression were covered in this chapter. Each algorithm is discussed in detail here, along with examples of how to use them. In addition, we learned more in-depth about linear and logistic regression by showing them with a Python application on real-world data.

Reinforcement Learning

Abstract: This chapter introduces the readers to a new concept of machine learning, other than supervised and unsupervised learning. This concept is popularly known as reinforcement learning. Reinforcement learning is a kind of machine learning algorithm, where the model learns itself based on surrounding behavior and new technique of rewarding. This chapter will gradually teach the readers about each concept for understanding reinforcement learning in-depth, alongside a basic application with Python. This chapter will look at the concepts to understand why it is getting so much attention these days. This serves as a beginner's guide to reinforcement learning. Reinforcement learning is undoubtedly one of the most visible study areas at the moment, with a promising future ahead of it, and its popularity is growing by the day.

Keywords: Machine Learning, Reinforcement Learning, Environment, Reward, Agent, Learning.

5.1. INTRODUCTION TO REINFORCEMENT LEARNING

Reinforcement learning (RL) is a component of machine learning that instructs an agent to select an action from its action space inside a given environment for maximizing rewards throughout time.

Its purpose is to teach the model how to conduct certain activities in a specific environment, leading to the discovery of the best actions to take in various scenarios to reach the ultimate objective. Using repeated trials to maximize a cumulative reward, the agent learns to attain a goal in an unpredictable and complicated environment. The agent uses a trial-and-error approach to find a solution to the problem, and the actions it takes, earn it either rewards or penalties. Reinforcement Learning is used to determine the optimum behavior or course to follow in a particular circumstance.

Let us now understand the concept in a more elaborative way, rather than just mentioning the terminologies. Reinforcement learning is a new technique of machine learning, not a very old concept. So, it is still under the research and development phase for utilizing its capacity at most.

5.1.1. Overview

As we know, reinforcement learning is part of a machine learning algorithm, which is pretty advanced. So, let us quickly overview the other algorithms and reinforcement learning, too.

Machine learning is a section of computer science dealing with the creation and development of algorithms that enable computers to learn from data sources, such as sensor data or databases. Detecting complicated patterns and making intelligent judgments based on data is a crucial focus of machine learning research.

- **Supervised Machine Learning:** Immediate feedback is available with supervised learning.
- **Unsupervised Machine Learning:** There is no feedback in unsupervised learning.
- **Reinforcement Learning:** Scalar feedback is delayed in reinforcement learning (a value called reward).

RL is about agents who must notice and react to their surroundings. This method combines traditional AI and machine learning methods. It is a complete problem-solving environment.

Let us consider the following examples:

1. A robot is cleaning a room while its battery is being recharged.
2. Soccer played by robots
3. How to invest in a stock market smartly.
4. Using rational agents to model the economy
5. Learning to fly a helicopter is a challenging task.
6. Planes are scheduled to arrive at their destinations.

5.1.1.1. Meaning of Reinforcement

The occurrence of an event in the context of a reaction increases the likelihood of the reaction occurring again in the same scenario.

The challenge that an agent encounters while learning behaviour in a dynamic environment through trial-and-error is known as reinforcement learning. It is a method of figuring out how to perform to maximize a monetary return.

It is neither a sort of neural network nor a substitute for neural networks. Instead, for the learning device, it is an orthogonal technique. Reinforcement learning provides feedback that evaluates the learner's performance without establishing criteria of accuracy in the form of behavioral objectives. For example, we can consider bicycle training.

5.1.1.2. How RL Differs from Supervised Learning

The model is trained with a training dataset that includes a correct response key in supervised learning. The decision is based on the initial input, which contains all the data necessary to train the computer. Because the judgments are distinct from one another, each one is represented by a label.

There is no response in reinforcement learning; thus, the reinforcement agent determines what to do to complete the job. Since the training dataset was unavailable, the agent had to rely on its own experience to learn. It all comes down to making judgments in a logical order. Simply put, the output depends on the current input state, and the following input depends on the output of the previous input. It labels the dependent decisions in order.

5.1.2. Element of Reinforcement Learning

Before getting deeper into the working model of reinforcement learning, let us first understand the key terminologies used in RL. Fig. (5.1) states the overall architecture of reinforcement learning. We observe two significant elements, agent and environment. In this diagram, we observe that these two blocks are connected with three arrows; the first one is the state, which connects the agent from the environment. The second one is action, which relates the environment from the agent and is supported by a policy. The third one is the reward, which connects the

agent from the environment [12]. Fig. (5.1) illustrates the overview of the working condition of reinforcement learning.

Agent: A sophisticated and intelligent program is referred to as an agent.

Environment: An external condition is an environment.

Policy: It defines an agent's conduct at a specific point in time. A policy is a relationship between states and actions. It can be in the form of lookup tables or a simple function.

Reward: In an RL problem, the reward function defines the aim. To attain this purpose, the policy is changed.

Value Function: A reward function explains what is desirable in the short term, but a value function describes what is beneficial over time. The value of a state is the total amount of reward an agent might expect to get in the future, starting from that state.

Environment Model: The process replicates the behavior of the environment using a model of the environment. It is used for planning, and if one knows its present state and action, one can forecast the next state and reward.

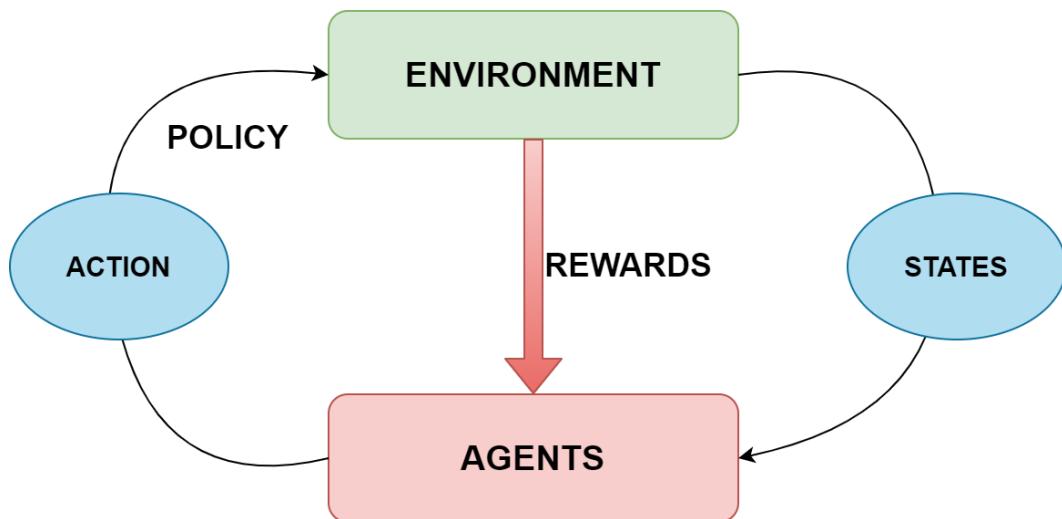


Fig. (5.1). Overview of the working condition of reinforcement learning.

5.2. ALGORITHMIC FRAMEWORK

5.2.1. Basic Steps of Reinforcement Learning

1. The agent monitors the input conditions.
1. Behavior is determined by the decision-making process (policy).
2. The action is executed.
3. Agents receive scalar rewards or enhancements to their environment.
4. State-action pair and rewards are reported.

❖ There are quite a few methods to solve the problem in reinforcement learning.

Strategies do not work with a value-based reinforcement learning approach. The best way to maximize the value function $V(S)$ is to take action.

Policy-based Approach: According to a policy-based approach, the actions taken in each state are based on policy features for maximum rewards in the future. There is no value function involved. Policy functions are deterministic and can generate the same action A in any state or probability theory where each action has a specific probability of occurrence.

Model-based Approach: The model-based approach creates a virtual model for each environment, and the agent learns to act in that particular environment. There is no single solution or algorithm for this approach, as the model varies from environment to environment.

Value-based: This method's primary purpose is to optimize a value function. An agent expects a long-term restoration of the existing states through a policy.

5.2.2. Types of Reinforcement Learning

When discussing reinforcement learning, it is to be noted that RL can be broadly categorized into two different types, namely positive reinforcement learning and negative reinforcement learning [13].

1. Positive Recommendation

When an event occurs due to a specific behavior, positive reinforcement occurs, increasing the strength and frequency of the behavior. It has a favorable effect on conduct. In addition, it increases the effectiveness of activity, maintaining change for a more extended amount of time. However, excess reinforcement might result in an overflow of states, lowering the final output.

2. Negative Reinforcement

The strengthening of a habit is referred to as negative reinforcement. In other words, if a negative situation is banned or avoided, it seeks to prevent the action from occurring again. Thus, it has a maximized behavior and performs in a reasonable minimum standard. However, it just restricts itself to satisfy a minimum standard of behavior.

5.2.3. Elements of Reinforcement Learning

❖ Currently, there are two primary models available for implementing reinforcement learning algorithms in real life. One is Markov's decision process, and another one is the Q-learning method. In this section, we will learn both of them in detail. Before getting into the detailed working structure of models of RL, let us get accustomed to a few essential terms used in RL. These terms help understand the working methodology of reinforcement learning.

❖ Reinforcement learning uses the evaluative feedback technique. There are two kinds of evaluative feedback techniques available.

▪ Purely Evaluative Feedback

- It assesses the effectiveness of the activity done.
- It is difficult to say if this is the best or worst course of action.
- The foundation of function optimization approaches.

▪ Purely Instructive Feedback

- It indicates the proper course of action to take, regardless of the activity taken.
For example, supervised learning.

❖ Associative & Non-associative Tasks**○ Associative**

- It is situation-specific.
- It helps in mapping from a circumstance to the optimum actions for that scenario.

○ Non-associative

- It is independent of the situation.
- There is no need to link various actions to distinct scenarios.
- When the job is fixed, the learner either tries to discover a single best action or tries to monitor the best action as it varies over time when the work is not fixed.

❖ Exploration and exploitation.**○ Greedy action (Exploitation)**

- Choosing the action with the highest assessed value.
- Greedy behavior is an example of exploitative behavior.

○ Non-greedy action (Exploration)

- This is an example of exploration since it allows us to assess the non-greedy action value better.

○ Greediest action

- The greediest activity is one that has the most significant expected return.

▪ **ϵ -greedy**

- Usually, the most greedy action is chosen.
- An action is chosen at random now and then with a bit of probability (ϵ). The action is chosen consistently, regardless of the action-value estimations.

▪ **ϵ -soft**

- The optimal action is chosen with probability $(1-\epsilon)$, and a random action is chosen uniformly the remainder of the time.

5.2.4. Models of Reinforcement Learning

As we mentioned in the last section, amid various models in reinforcement learning, the most widely used models are Markov's decision process [12] and Q-learning [14], which are defined below:

5.2.4.1. Markov's Decision Process (Mdp)

In reinforcement learning, Markov Decision Process models are mathematical frameworks for mapping solutions. The group of variables includes S , which is for a set of finite states, A stands for a set of feasible actions in each state, R stands for reward, M is for a model, and π stands for policy [12]. The consequence of deploying an action to a state is determined by the present action and state, not by past actions or states.

$$\text{State} = S$$

$$\text{Model} = M(S, a, S')$$

$$\text{Action} = A(S), A$$

$$\text{Reward} = R(S), R(S, a), R(S, a, S')$$

$$\text{Policy} = \pi(S) \rightarrow a$$

$$= \pi^*$$

Let us see an example to understand MDP. For example, let us consider a fair dice rolling game. The dice have fair corners and six different faces, numbers ranging from 1 to 6. Fig. (5.2) illustrates an example showing the dice game using the MDP model. The condition of the games lies is:

1. You have the option to continue or stop each round.
2. If you resign, you will be awarded 10 points, and the game will be over.
3. If you keep going, you'll get 6 points and 6-sided dice. The game is over if the dice lands on a 1 or 2. If not, the game moves on to the following round.
4. There's an obvious trade-off to be made here. For example, we can trade a 4 point guaranteed gain for the chance to roll dice and advance to the next round.

As we know, a Markov Decision Process is formalized as $\mathbf{m} = (\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}, \gamma)$, where:

\mathbf{S} = It stands for the set of all states.

\mathbf{A} = It represents the collection of actions that can be taken.

\mathbf{P} = The transition probabilities are denoted by the letter P.

\mathbf{R} = It stands for rewards.

γ = The discount factor is denoted as gamma.

To apply MDP in reinforcement learning for this application, let us define the $\mathbf{S}, \mathbf{A}, \mathbf{P}, \mathbf{R}$, according to the conditions of the game.

State: A state is a possible state for the agent (decision-maker).

Agent: The agent might be in the game or out of the game in the dice game.

Action: A movement that the agent may choose is referred to as an action. It transports the agent between states while imposing various penalties and incentives.

Transitions: Given an action a , transition probabilities explain the chances of ending up in a state S' . These are frequently labeled as $P(S, a, S')$ functions that output the likelihood of ending up in S' , given the current state S and action A .

Reward: Depending on the action, rewards are offered. Continuing the game earns you 6 points, while leaving earns you 10 points. It is necessary to maximize the 'overall' return.

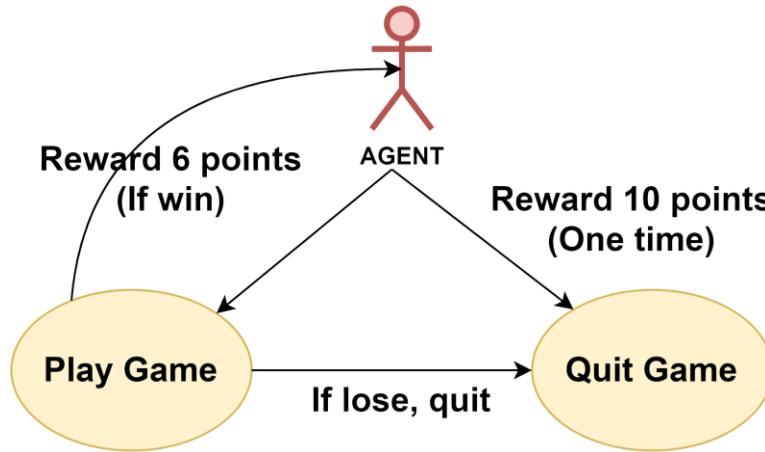


Fig. (5.2). Example showing the dice game using the MDP model.

Policy: The primary purpose is to develop a policy, sometimes abbreviated as π , that produces the best long-term return. Policies are just a mapping of each state S to an action distribution. With a specific probability for each state S , the agent should do action A .

❖ Bellman's Equation

To understand MDP, Bellman's equation plays a vital role. It is the core part of Markov's decision process model. It lays forth a framework for calculating the optimal projected reward in a given state by addressing the question, "What is the maximum reward an agent may earn if they do the best action now and in the future?"

Although Bellman's equation is a bit complicated, a widely used simpler version is available to understand. This equation can be framed as:

$$V(S) = \max_a R(S, a) + \gamma V(S')$$

Where,

$V(S)$ stands for the expected return during the current state;

$R(S, a)$ stands for the expected reward for action 'a' at state 'S';

$\gamma V(S')$ stands for discount factor (γ) times the return for next state;

\max_a stands for maximum return for any action 's'.

The importance of γ should be noted, ranging from zero to one ($0 \leq \gamma \leq 1$), when calculating the best reward. The $V(S')$ term is thoroughly wiped out when $\gamma = 0$, and the model solely cares about the immediate reward. If γ is adjusted to 1, on the other hand, the model influences prospective future benefits equally to immediate rewards. Thus, the best γ value is generally between 0 and 1, with the value of further-out rewards declining with the gap.

Let's utilize the Bellman formula to figure out how much money we could win in the dice game. Because we have two options, our enlarged equation will be max. (the reward of choice 1 vs. the reward of choice 2). The choice is to withdraw from the game, eventually giving a one-time reward of 10 points.

However, choice 2 results in a 6 points reward as well as a $\frac{2}{3}$ probability of progressing to the following round, when the option can be made again (calculating by expected return). In front of terms, we add a discount factor γ to indicate the calculation of S' . Even with a maximum $\gamma = 1$, the following equation is recursive, but it will finally converge to one value since the value of the following iteration drops by $\frac{2}{3}$.

$$\max(10, 6 + \gamma \frac{2}{3} (\max \left(10, 6 + \gamma \frac{2}{3} \left(\max \left(10, 6 + \gamma \frac{2}{3} \dots \right) \dots \right) \right)))$$

Either we can withdraw from the game and receive an extra 10 points in expected value, or we may stay and receive an extra 6 points in anticipated value at each stage. The predicted value increases by two-thirds for each subsequent round because there is a two-thirds chance of continuing even if the agent decides to stay.

❖ Few Concepts of MDP

1. Value Functions

a. State Value Function: $V^\pi(S)$

- i. Under a policy, the value function indicated as $V(\mathbf{S})$ expresses how good a state is for an agent. In other words, beginning from the current condition under policy π , what is the average benefit that the agent will receive?
- ii. The expected return when starting in state S and following the policy π .
- iii. As previously stated, the policy is mapping the probability of doing each conceivable action in each state ($\pi(a/s)$). When a policy informs precisely what to do in each condition and does not provide probabilities, it is said to be deterministic.
- iv. It should now be self-evident that maximizing the value function for each state would yield "the optimal policy (π^*)". The following is the equation of optimal policy:

$$\pi^* = \arg \max V^\pi(S) \forall S \in \mathcal{S}$$

b. State-action Value Function: $Q^\pi(S,a)$

- i. A state-action value function, commonly known as the q-value, does this.
- ii. The expected return when starting in state S , performing action A , and following policy π .

❖ Dynamic Programming

Dynamic Programming was coined by Richard Bellman and is used to solve issues that can be broken down into subproblems. With the crucial assumption that the features of the environment are known, dynamic programming can assist in identifying optimum solutions to industry planning challenges. It is a significant step to begin learning about RL algorithms that can tackle more complicated tasks.

We can effectively identify an optimum policy for the agent to follow given the complete model and specifications of the environment (MDP). It consists of two significant steps:

1. Solving the big problem by breaking it down into smaller subproblems.
2. Subproblem solutions are cached or saved for later use to obtain the best solution to the problem.

To solve an MDP, the solution must have the following components:

- Determine the effectiveness of an arbitrary policy.
- Determine the best policy for the given MDP.

There are two popular methods for Dynamic programming, which are as follows:

1. Policy Evaluation: It is an iterative process, computing value function for a given policy (in case of prediction problem): computing V^π from π . The question of how excellent a policy is answered by policy evaluation.

Bellman equations define a system of 'n' equations, which could solve the problem. However, it will use the iterative version of the equation, as stated above. It will start with any arbitrary value function (V_0) and continue to iterate until the V_k converges.

2. Policy Improvement: It is basically a straightforward computation for finding an improved policy for a given value function: improving π based on V^π .

We determined the value function V for an arbitrary policy π using policy evaluation. We are well aware of the effectiveness of our existing policy. For specific states, we want to know the consequences of doing an action that isn't policy-related. Let's imagine we choose an action A in state S and then stick to the original policy π .

$$\begin{aligned}\pi^*(S) &= \arg \max Q^\pi(S, a) \\ &= \arg \max \sum_{S'} P_{SS'}^a [r_{SS'}^a + \gamma V^\pi(S')]\end{aligned}$$

- The π^* either strictly better than π or π^* is optimal (if $\pi = \pi^*$).

3. Policy Iteration: We may then compute V^π to enhance the policy further to $V^{\pi''}$ after it has been improved using V to provide a better policy π^* . Iterations are repeated to get close to the real value function for a particular policy π . Policy iteration is the process of improving a policy, as mentioned in the policy improvement section.

As a result, the new policy will almost certainly be better than the old one, given enough iterations. It will eventually return the best policy. This sounds excellent, but there is a catch; each iteration of policy iteration includes another iteration of policy review, which might necessitate repeated sweeps over all states.

5.2.4.2. *Q-learning*

In reinforcement learning, Q-learning is a model-free, value-based strategy for delivering information to inform an agent's action. It is based on the concept of updating Q values, which demonstrates the value of performing action A in state S [14]. The essential feature of the Q-learning algorithm is the value updating rule.

The overall steps of Q-learning are as follows:

1. Iterate until the goal is achieved:
 - a. Initialize the Q-table
 - b. Select an action
 - c. Perform that chosen action
 - d. Measure the reward
 - e. Update the Q-table

Instead of identifying the potential worth of the state being moved to, Q-Learning proposes analyzing the quality of an activity made to go to that state. We don't know about probabilities in Q-learning because they aren't explicitly described in the model. Instead, through interacting with the surroundings, the model must learn this and the environment on its own.

As a result, Q-learning is appropriate for situations in which explicit probabilities and values are unknown. However, you may not need to employ Q-learning if they are already known.

$$V(S) = \max(R(S, a) + \gamma \sum S' P(S, a, S') V(S'))$$

We know this equation returns a value of going to a specific state, considering the environment's stochasticity. Suppose we add the concept of evaluating the quality of activities taken to reach a specific state S' . To determine $Q(S, a)$, i.e., the cumulative quality of its possible actions, we must first deconstruct the formula. When we remove the `max()` method, we obtain the following:

$$R(S, a) + \gamma \sum S' (P(S, a, S') V(S'))$$

Generally, we include all potential actions and states in the equation that yields $V(S)$, and then we take the most prominent value induced by doing a given action. Thus, the value footprint generated by the calculation above is for only one possible action. In reality, we might consider it to be the action's quality.

$$Q(S, a) = R(S, a) + \gamma \sum S' (P(S, a, S') V(S'))$$

We'll make a minor tweak to the previous equation now that we have an equation to assess the quality of a specific action. Now, we may state that $V(S)$ is the highest of all possible Q values (S, a). Let's use this to our equation and replace $V(S')$ with a function of $Q()$.

$$Q(s, a) = R(s, a) + \gamma \sum s' (P(s, a, s') \max Q(s', a'))$$

We just need to compute one function, Q , and $R(S, a)$ is a quantified measure that generates rewards for going to a specific state. Q-values are the quality of the actions.

5.2.4.2.1. Temporal Difference

The temporal difference will assist the computer in computing Q-values in response to environmental changes over time. Think about where the computer is now in the indicated state and wishes to go to the higher state. It's worth noting that the robot (automated system) already understands the Q-value of doing the action while going to the higher state.

We know that the environment is stochastic. Therefore, the robot's reward after shifting to the top state may differ from previously seen. We use the same procedure to recalculate the new $Q(S, a)$ and deduct the previously calculated $Q(S, a)$ from it.

$$TD(a, S) = NEW_Q(S, a) - Q(S, a)$$

The above-mentioned equation calculates the temporal difference in Q-values, which aids in capturing the random variations that the environment may impose. The 'new' $Q(S, a)$ has been modified as follows:

$$Q_t(S, a) = Q_{t-1}(S, a) + \alpha TD_t(a, S)$$

In the above equation, if we replace $TD_t(a, S)$ with its complete mathematical form, then the final equation would be:

$$Q_t(S, a) = Q_{t-1}(S, a) + \alpha(R(S, a) + \gamma \max Q(S', a') - Q_{t-1}(S, a))$$

In the above equation, the α is called the learning rate. It maintains the adaptability of the computer to random environmental changes. $Q_t(S, a)$ is the Q-value at the current state, and $Q_{t-1}(S, a)$ is the Q-value at an earlier state.

5.2.4.3. Hands-on Example of Reinforcement Learning

Let's take an example of a self-moving robot, which is being placed in a room of blocks and advised to reach the goal (Fig. 5.3).

It is instructed to move in only four directions: up, down, left, and right. Corner movements are not allowed.

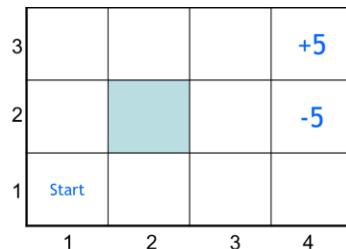


Fig. (5.3). Example showing the goal and path of a reinforcement learning example.

It will get a reward (R) of $+5$ at the block's position [4,3], and the reward will be subtracted with -5 values if it reaches the block position [4,2]. For every step it moves, the robot will be given a reward of -0.04 , *i.e.*, the robot's goal is to reach the target and in fewer possible steps. The blue block at position [2,2] is a 'blocked' block, which needs to be avoided. The start position is at [1,1].

Our target is to find a suitable solution for the robot to reach the target, earn the maximum reward, and maintain the states, actions, and rewards. As we know, in reinforcement learning, the actions are stochastics, and the policy is to map a path from each state to action.

The robot will perform some probable steps to move through the room with blocks for earning the reward. However, the optimal steps may be similar to the steps mentioned below. The figure below shows the robot's movement across the room for 'step-wise reward of -0.04 (Fig. 5.4).

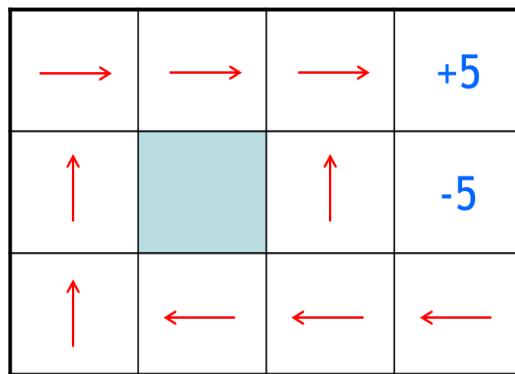


Fig. (5.4). The probable step for reward -0.04 .

- For a step-wise reward = -0.04

As we know, the optimal solution is not easy to achieve. Now, let us see some probable steps if we keep changing the 'Step-wise' reward (Fig. 5.5).

- For a step-wise reward = -2

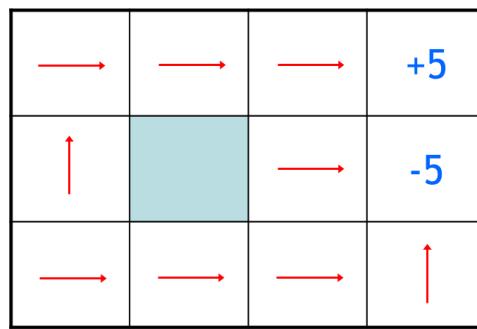


Fig. (5.5). Probable steps for a reward of -2.

- For a step-wise reward = -0.1 (Fig. 5.6).

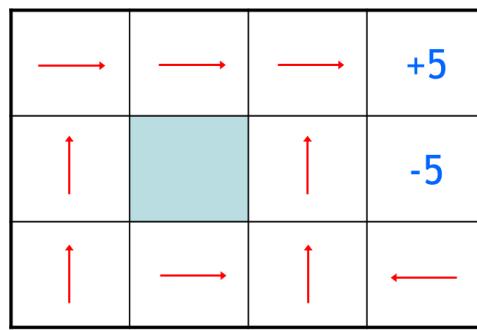


Fig. (5.6). Probable steps of a reward of -0.1.

- For a step-wise reward = -0.01 (Fig. 5.7).

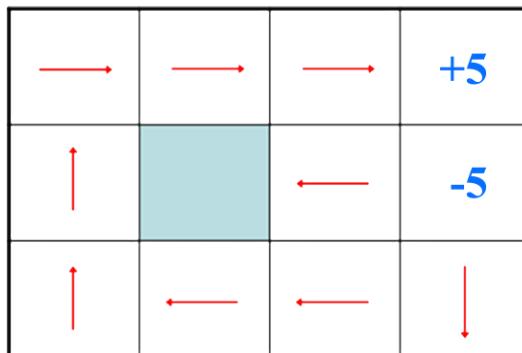


Fig. (5.7). Probable steps for a reward of -0.01.

- For a step-wise reward = +0.01 (Fig. 5.8).

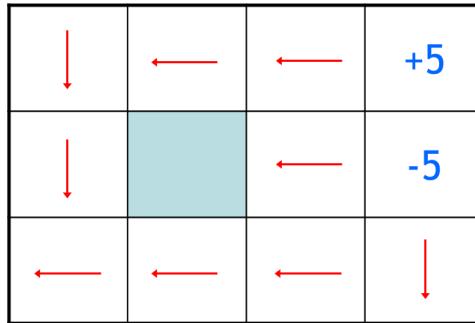


Fig. (5.8). Probable steps for a reward of +0.01.

5.3. REAL-WORLD EXAMPLES OF A REINFORCEMENT LEARNING TASK

Till now, we learned the basics of reinforcement learning and its detailed architecture. After understanding the working principle of RL, we are here to see some real-life applications of RL, which will be mentioned one by one, in short. The following are the applications that we can build, develop, and improve.

1. Controlling A Walking Robot

In this application, the robot can be taught to move more effectively by optimizing its strategy based on its rewards.

Agent: Walking robot control program.

Environment: The real physical world around the robot.

Action: One of four movements (1) Moving forward, (2) Moving backward, (3) Turning left and (4) Turning Right.

Reward: There is an addition of reward value when it reaches its goal. If it moves in the wrong direction or halt or fall, it will lose the point of subtraction of reward.

2. Developing a Customized Learning System

This application has the potential to increase the value of a customized class system. For example, more effective learning can help the user, and more successful advertising can benefit the system.

Agent: In an online learning catalog, the program determines what to display next.

Environment: The learning system is the environment.

Action: Action is now showing a new class video as well as advertising.

Reward: If the user decides to click the class video, the reward is positive; if the user decides to click the advertisement, the reward is more extensive; and if the user decides to leave, the reward is negative.

3. Choosing an Advertisement's Location on a Web Page

In this application, the agent in this scenario monitors the surroundings and determines their present state. The status can indicate how many adverts are currently on the page and whether or not additional adverts can be added. The agent then selects one of the three options for each phase. It can design an effective policy if it is designed to get positive incentives whenever revenue increases and negative incentives if revenue decreases.

Agent: The program that determines how many advertisements are suitable for a particular page.

Environment: The website.

Action: One of the three options: (1) adding a new advertisement to the page, (2) deleting an advertisement from the page, (3) neither adding nor deleting an advertisement from the page.

Reward: When revenue grows, it is added. However, when revenue decreases, it is subtracted.

Let us now dig into some other real-world applications of reinforcement learning, which are given below:

4. Application in Healthcare

Patients in healthcare can benefit from policies learned through reinforcement learning systems. Without prior knowledge of the mathematical model of biological systems, RL can develop optimum policies based on previous experiences. It makes this method more suitable in healthcare than other control-based systems. Dynamic treatment regimens (DTRs) in chronic disease or critical care, automated medical diagnostics, and other broad fields are examples of RL in healthcare.

A series of clinical observations and evaluations of a patient is used as the input in DTRs. The treatment choices at each step are the outcomes.

5. Application in News Recommendations

As we know that users' tastes vary regularly, offering news to people based on ratings and likes might rapidly become outdated. The reinforcement learning system may track the reader's return behaviors using reinforcement learning. Obtaining news features, reader features, context features, and reader news features would be required to build such a system. Content, headline, and publisher are just a few examples of news features. The reader's interaction with the material, such as clicks and shares, is referred to as reader features. News elements such as time and freshness of the news are examples of context characteristics. Following that, a reward is determined depending on the user's actions.

6. Reinforcement Learning in Automated Self-driving Cars

There are several factors to consider with self-driving automobiles, including speed limitations in various locations, drivable zones, and avoiding crashes, to name a few.

Trajectory optimization, motion planning, dynamic pathing, controller optimization, and scenario-based learning policies for highways are some of the autonomous driving activities, where reinforcement learning might be used. Learning automated parking regulations, for example, can help with parking. Q-Learning may be used to change lanes, and overtaking can be done by learning an overtaking strategy while avoiding collisions and keeping a constant speed.

7. Applications in Stock Trading and Finance

Forecasting future sales and stock prices may both be done with supervised time series models. On the other hand, these models do not determine what to do at a given stock price. A reinforcement learning agent can select whether to keep, purchase, or sell a task. To guarantee that the RL model is working correctly, it is assessed using market benchmark criteria.

Unlike prior techniques, which required analysts to make every choice, automation ensures uniformity throughout the process. Every financial transaction's loss or profit is used to calculate the reward function.

8. Application in Gaming

Let's take a look at a game program, especially AlphaGo Zero. AlphaGo Zero was able to learn the game of Go from the ground up using reinforcement learning. It figured out how to play against itself. Alpha Go Zero exceeded the version of Alpha Go known as Master, which had beat world number one Ke Jie after 40 days of self-training. It had a single neural network and only utilized black and white stones from the board as input characteristics. A basic tree search based on a single neural network is utilized to evaluate location and sample movements without employing Monte Carlo rollouts.

5.4. ADVANTAGES AND DISADVANTAGES OF REINFORCEMENT LEARNING

5.4.1. Advantages

- It assists you in determining which situations require action.
- It aids you in determining which activity delivers the most significant benefit over time.
- It also includes a reward function for the learning agent.
- It also enables it to determine the most efficient means of collecting significant rewards.

5.4.2. Disadvantages

- Design of the feature or reward, which should be highly involved.
- A variety of factors can influence the pace with which it learns.
- Partially observable surroundings are possible in realistic settings.
- Too much reinforcement might result in an overabundance of states, lowering the quality of the output.
- Non-stationary situations are possible in realistic settings.

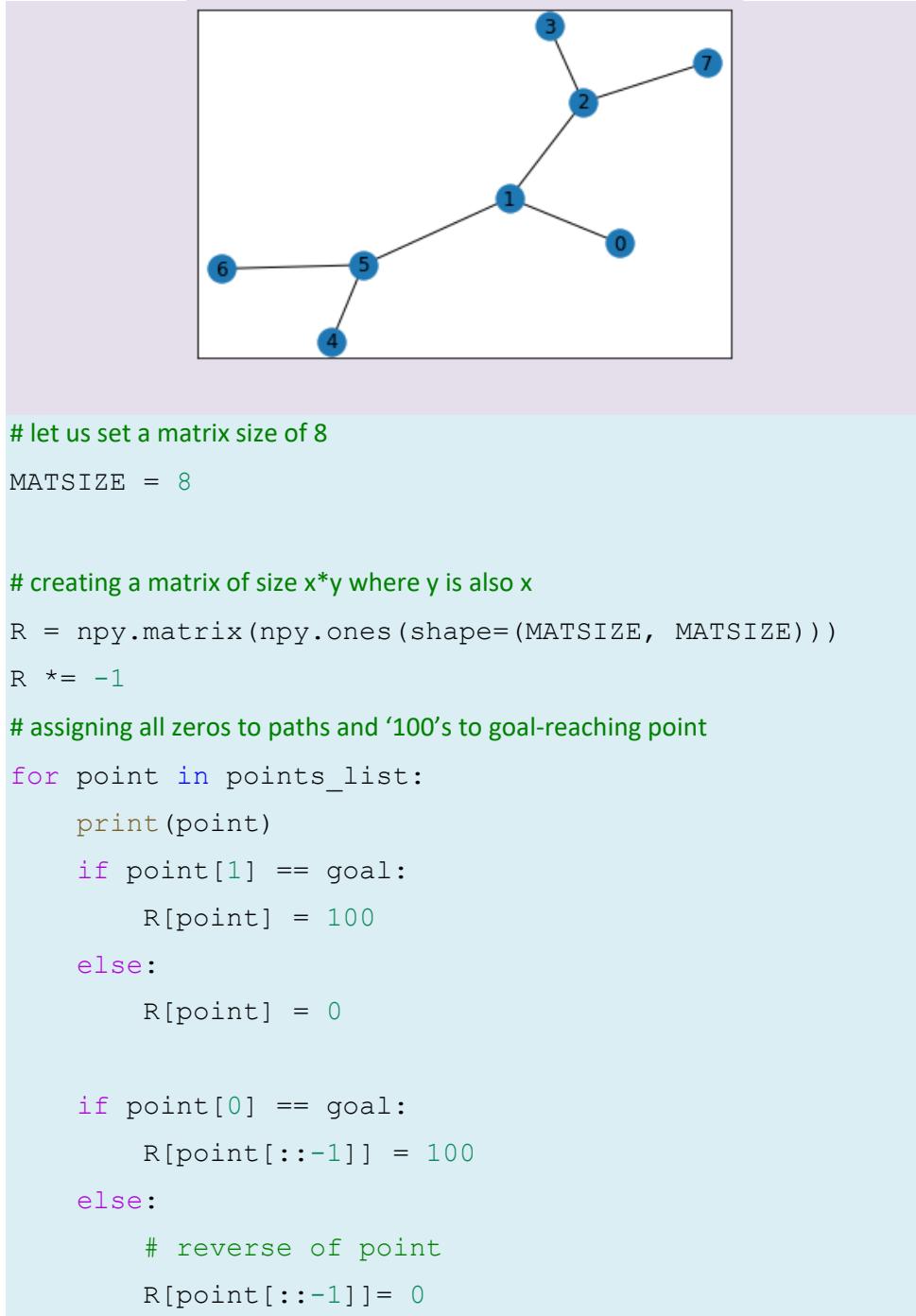
5.5. PROGRAMMING APPROACH FOR REINFORCEMENT LEARNING

```
#Reinforcement learning: simple python q-learning

import numpy as npy
import pylab as plt
# mapping      the cell to      each      other, adding
a circular cell to the goal-point

points_list = [(0,1), (1,5), (5,6), (5,4), (1,2), (2,3),
), (2,7)]
goal = 7

import networkx as nx
G=nx.Graph()
G.add_edges_from(points_list)
pos = nx.spring_layout(G)
nx.draw_networkx_nodes(G,pos)
nx.draw_networkx_edges(G,pos)
nx.draw_networkx_labels(G,pos)
plt.show()
```



```

# adding the goal-point - round trip
R[goal,goal]= 100

(0, 1)
(1, 5)
(5, 6)
(5, 4)
(1, 2)
(2, 3)
(2, 7)

R
matrix([[ -1.,   0.,  -1.,  -1.,  -1.,  -1.,  -1.,  -1.],
       [  0.,  -1.,   0.,  -1.,  -1.,   0.,  -1.,  -1.],
       [ -1.,   0.,  -1.,   0.,  -1.,  -1.,  -1.,  100.],
       [ -1.,  -1.,   0.,  -1.,  -1.,  -1.,  -1.,  -1.],
       [ -1.,  -1.,  -1.,  -1.,  -1.,   0.,  -1.,  -1.],
       [ -1.,   0.,  -1.,  -1.,   0.,  -1.,   0.,  -1.],
       [ -1.,  -1.,  -1.,  -1.,  -1.,   0.,  -1.,  -1.],
       [ -1.,  -1.,   0.,  -1.,  -1.,  -1.,  -1.,  100.]))

Q = npy.matrix(npy.zeros([MATSIZE,MATSIZE])) 

# learning parameter
gamma = 0.8

StateInit = 1

```

```
def actAvail(state):
    stateCurr_row = R[state,]
    av_act = npy.where(stateCurr_row >= 0) [1]
    return av_act

totalAvailAct = actAvail(StateInit)

def nextActSample(actAvail_range):
    nextAct = int(npy.random.choice(totalAvailAct, 1))
    return nextAct

action = nextActSample(totalAvailAct)

def update(stateCurr, action, gamma):

    IndMax = npy.where(Q[action,] == npy.max(Q[action,]))
    ) [1]

    if IndMax.shape[0] > 1:
        IndMax = int(npy.random.choice(IndMax, size = 1))
    )
    else:
        IndMax = int(IndMax)
    ValMax = Q[action, IndMax]

    Q[stateCurr, action] = R[stateCurr, action] + gamma
    * ValMax
```

```
    print('ValMax', R[stateCurr, action] + gamma * ValMa
x)

    if (npy.max(Q) > 0):
        return(npy.sum(Q/npy.max(Q)*100))
    else:
        return (0)

update(StateInit, action, gamma)

ValMax 0.0
0
# Training
scores = []
for i in range(700):
    stateCurr = npy.random.randint(0, int(Q.shape[0]))
    totalAvailAct = actAvail(stateCurr)
    action = nextActSample(totalAvailAct)
    score = update(stateCurr,action,gamma)
    scores.append(score)
    print ('Score:', str(score))

print("Showing the trained 'Q' matrix:")
print(Q/npy.max(Q)*100)

# Testing
stateCurr = 0
steps = [stateCurr]
```

```
while stateCurr != 7:

    IndNextStep = npy.where(Q[stateCurr,] == npy.max(Q
[stateCurr,]))[1]

    if IndNextStep.shape[0] > 1:
        IndNextStep = int(npy.random.choice(IndNextStep,
size = 1))
    else:
        IndNextStep = int(IndNextStep)

    steps.append(IndNextStep)
    stateCurr = IndNextStep

print("Showing the most efficient path:")
print(steps)

plt.plot(scores)
plt.show()

ValMax 0.0
Score: 0
ValMax 0.0
Score: 0
ValMax 0.0
Score: 0
```

```
...
ValMax 0.0
Score: 100.0
ValMax 0.0
Score: 100.0
ValMax 0.0
Score: 100.0
...
ValMax 195.2000000000002
Score: 253.77049180327867
ValMax 195.2000000000002
Score: 333.7704918032787
ValMax 195.2000000000002
Score: 413.7704918032787
ValMax 0.0
Score: 413.7704918032787
ValMax 195.2000000000002
Score: 413.7704918032787
ValMax 195.2000000000002
Score: 413.7704918032787
...
ValMax 220.1231360000004
Score: 793.3629135606895
ValMax 156.1231360000004
Score: 793.3629135606895
ValMax 156.1231360000004
Score: 793.3629135606895
ValMax 320.0985088000005
```

```
Score: 804.5956028896065
ValMax 156.12313600000004
Score: 804.5956028896065
ValMax 320.09850880000005
Score: 815.8282922185235
...
ValMax 215.09043650560008
Score: 903.7658383345106
ValMax 215.09043650560008
Score: 903.7658383345106
ValMax 359.0904365056001
Score: 908.778277635306
ValMax 215.09043650560008
Score: 908.778277635306
ValMax 359.0904365056001
Score: 908.778277635306
...
```

```
ValMax 319.75851256065664
Score: 982.2907658174032
ValMax 319.75851256065664
Score: 982.2907658174032
ValMax 319.75851256065664
Score: 982.2907658174032
ValMax 399.95948513068856
Score: 982.2907658174032
ValMax 499.9493564133607
```

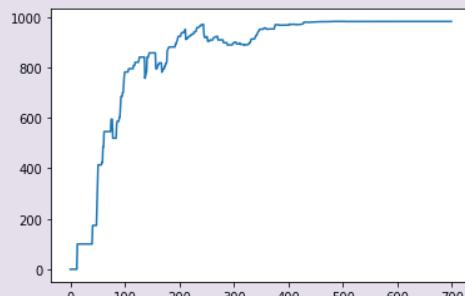
```

Score: 982.2932982532384
ValMax 319.75851256065664
Score: 982.2932982532384
Trained Q matrix:
[[ 0.           63.95818066   0.           0.           0.
   0.           0.           0.           ] ]
[ 51.16654452   0.           79.94772582   0.           0.
  51.16654452   0.           0.           ] ]
[ 0.           63.95818066   0.           63.99065531   0.
   0.           0.           100.          ] ]
[ 0.           0.           80.           0.           0.
   0.           0.           0.           ] ]
[ 0.           0.           0.           0.           0.
  51.16654452   0.           0.           ] ]
[ 0.           63.95818066   0.           0.           40.93323562
   0.           40.93323562   0.           ] ]
[ 0.           0.           0.           0.           0.
  51.16654452   0.           0.           ] ]
[ 0.           0.           79.94772582   0.           0.
   0.           0.           100.          ] ]

```

Most efficient path:

```
[0, 1, 2, 7]
```



CONCLUDING REMARKS

Reinforcement learning (RL) is a machine learning subfield similar to supervised or unsupervised learning but differs in numerous aspects. It is more difficult to apply to real-world business scenarios since it requires simulated data and surroundings. On the other hand, RL technology is promising since its learning process is inherent to sequential decision-making settings.

CHAPTER 6

Deep Learning: A New Approach to Machine Learning

Abstract: The chapter will introduce the readers to the latest state-of-the-art deep learning algorithms from scratch. Deep learning is a modern field of machine learning capable of understanding the underlining patterns in the data on its own and identifying the nature of the data. This chapter will travel through all the algorithms, from basic neural network structure to advanced neural networks, such as convolution neural networks and recurrent neural networks. It covers artificial neural networks, perceptron learning algorithms, convolution neural networks, recurrent neural networks, long short term memory, and essential concepts such as backpropagation, gradient descent, activation functions, and optimizations. With the hands-on example and Pythonic approach to real-world applications, this chapter will enhance the readers' knowledge of advanced technologies.

Keywords: Deep Learning, Neural Network, Perceptron Learning Algorithm, MLP, ANN, Convolutional Neural Network, Random Forest, K-nearest Neighbor, Naïve Bayes Classification, Support Vector Machine.

6.1. INTRODUCTION TO NEURAL NETWORK

Deep learning is a subclass of machine learning that uses neural networks to learn from unstructured or unlabeled data in a manner comparable to that of the human brain. Giving the system vast data will eventually learn to comprehend it and respond in meaningful ways.

Deep learning is a kind of machine learning that learns to represent the world as a layered hierarchy of concepts. Each concept is defined in more straightforward concepts and more abstract representations calculated in less abstract representations. Each concept is represented in connection to other concepts. This provides it with an ample amount of power and flexibility.

We basically move to deep learning for getting the following:

- Deep learning works well with unstructured data.

- Deep learning can operate with structured and unstructured data, whereas machine learning works best with vast quantities of structured data.
- It can handle the complex operations.
- Machine learning algorithms are unable to execute complicated processes, but deep learning algorithms can.
- It can do the features engineering on its own.
- Machine learning methods detect patterns from labeled sample data, whereas deep learning algorithms receive a considerable volume of data as input and analyze it to extract characteristics from an object.
- Finally, it achieves better performance compared to machine learning algorithms.
- Machine learning algorithms' performance degrades as the number of data increases. Thus we require deep learning to keep the model running.

6.1.1. Architecture of Deep Learning

When we discuss deep learning architecture, we need to understand its working principle. As we know, deep learning is capable of doing both feature engineering and classification. It identifies the underlining pattern in the data to categorize it to a specific class [15].

A deep neural network (DNN) comprises several layers, transforming the input data into more abstract representations (for example, edge - nose - face). To generate predictions, the output layer integrates such features.

It's a sophisticated neural network (having multiple hidden layers in between input and output layers). Non-linear connections can be modeled and processed by them.

Deep-learning architectures like deep neural networks, deep belief networks, neural graph networks, recurrent neural networks, and convolutional neural networks are employed in areas like natural language processing (NLP), machine translation, computer vision, bioinformatics, speech recognition, drug design, medical image analysis, material inspection, and board game programs. In deep learning, the term "deep" refers to the usage of several layers in the network. Deep learning is a more

recent form with an infinite number of layers of fixed size, allowing for practical application and optimization while maintaining theoretical universality under moderate circumstances. In terms of efficiency, trainability, and understandability, deep learning layers are also heterogeneous and depart considerably from physiologically informed connectionist models, hence the "structured" component.

Like machine learning, deep learning can be of two types, supervised and unsupervised learning. Various algorithms can be listed in a supervised learning way, such as convolutional neural networks, recurrent neural networks, and various forms. The popular deep learning algorithms in the unsupervised part are neural-network-based self-organizing map, restricted Boltzmann machine, and autoencoders. The concept of supervised and unsupervised remains the same for both operations, as in machine learning algorithms.

6.1.2. Working Principle of Deep Learning

The working principle of deep learning algorithms lies in the architectural design of the model. It also depends on the kind of deep learning model, such as supervised and unsupervised. However, the basic working principle or algorithm can be framed in the following way:

1. First, to acquire the appropriate answer, we must identify the actual problem, which must be comprehended. The practicality of deep learning should also be evaluated.
2. Second, we must determine the pertinent facts that must relate to the actual situation and be adequately prepared.
3. Third, select a suitable deep learning algorithm.
4. Fourth, during training the dataset, an algorithm should be applied.
5. Fifth, the dataset should be subjected to final testing.

The recursive definition of an algorithm is as follows:

Step 1: Choose the initial weights at random.

Step 2: Apply the inputs to the network for each training pattern.

Step 3: Calculate the output for each neuron from the input layer to the output layer, passing through the hidden layer.

Step 4: Calculate the output error as follows:

Step 5: To compute error signals for pre-output layers, use the output error.

$$\text{Error}_b = \text{Output}_b(1 - \text{Output}_b) (\text{Target}_b * \text{Output}_b)$$

Step 6: Calculate weight adjustments using the error signals.

Step 7: Make the necessary weight changes.

$$\text{Weight}_{ab+} = \text{Weight}_{ab} + (\text{Error}_b * \text{Output}_a)$$

Where, Weight_{ab+} represents the new weight, Weight_{ab} represents the initial weight, and $\text{Output}_b(1 - \text{Output}_b)$ represents the sigmoid function.

The deep learning architecture runs mainly in two parts, the training phase and the testing phase. However, most of the operation is done in the training phase, which is described below.

6.1.2.1. Training Phase

The network's weights are individually specified and serve as the foundation for linking inputs to outputs. Training is the process of defining these weights and tuning them to translate the input to output with an acceptable degree of error across many iterations.

Back-propagation algorithms are widely used in multilayer networks to modify the network's weights. This is done by selecting a sample from the training set, applying the input, computing the output, and comparing the results to the predicted results (the forward pass).

The error is the gap between the actual and intended output. This error is used to modify the network's weights, starting with the output layer and working backward to the input layer. This back-propagation of the error to change the weights in the network reduces the error for the complete training set across many training samples.

6.1.3. Types of Deep Learning

The deep learning architecture majorly works in three primary categories: feedforward neural network, recurrent network, and symmetrically connected neural networks [16].

1. Feedforward Neural Network: This is the most popular form of neural network in practical applications. The input is the first layer, and the output is the final. We refer to neural networks with more than one hidden layer as "deep" neural networks. They perform several modifications to alter the similarity between cases. Each layer's neurons' activity is a non-linear dependence of the activities in the layer below.

2. Recurrent Neural Network: In their connection graph, they have directed cycles. As a result, following the arrows might occasionally lead you back to where you started. They can have complex dynamics, which can make training them quite challenging. They have a more biologically realistic feel about them.

3. Symmetrically Connected Neural Network: These are similar to recurrent networks, except the unit connections are symmetrical. Recurrent networks are significantly more challenging to examine than symmetric networks. As they follow an energy function, they are likewise limited in what they can achieve. "Hopfield Nets" are symmetrically linked nets with no hidden units. Boltzmann machines are symmetrically linked networks containing hidden components.

Based on these architectures, deep learning researchers have developed several state-of-the-art algorithms to date. There are eight kinds of deep learning algorithms among these algorithms, which are most prominent [16].

1. Perceptrons Learning

Perceptrons are the earliest generation of neural networks, which are nothing more than computer representations of neurons. Frank Rosenblatt made him famous in the early 1960s. They appear to have a sophisticated learning algorithm and have made several significant claims about what they can learn. The book "Perceptrons," written by Minsky and Papert in 1969, described what they could achieve and demonstrated their limitations. Many individuals think that all neural network models have these drawbacks. On the other hand, Perceptron learning is still

frequently employed today for problems involving large aspect vectors with millions of features.

In the standard paradigm of statistical system recognition, we first convert the source input vector into the vector of the characteristic process. We use a handwritten program based on common sense to define features. Next, we learn how to weigh each feature function to obtain a single scale. If this size is more significant than a certain threshold, we determine that the input vector is a positive example of the target class. The standard perceptron structure follows a feed-forward model, sending input to neurons, processing, and outputting. The network reads from bottom to top: input comes from the bottom, and output comes from the top.

2. Convolutional Neural Networks

Yann LeCun and his coworkers created LeNet, a very good recognizer for handwritten numbers, in 1998. In a feedforward network, they introduced several hidden layers, mapping neuronal units within each layer, pooling the output layer, and few other modifications. This new network is capable of doing a lot more tasks than just classification. Convolutional neural networks are the name given to them afterward.

Convolutional neural networks may be used to recognize everything from handwritten digits to three-dimensional objects. Recognizing real things in color images acquired from the internet, on the other hand, is far more complicated than recognizing handwritten numbers. Each image has a hundred times as many classes, pixels, two-dimensional images of three-dimensional sceneries, crowded scenes needing segmentation, and numerous objects.

3. Recurrent Neural Network

A basic review of sequence modeling is required to comprehend recurrent neural networks. We frequently wish to transform an input sequence into an output sequence in a different domain when using machine learning on sequences, for example, turning a sequence of sound notes into a sequence of word identities. When there isn't a separate goal sequence, we can predict the following word in the input sequence to receive a training signal. The input sequence is advanced by one step in the predicted output sequence. Trying to predict one pixel among other

pixels in an image or one patch of an image from the rest of the image appears much more natural.

Recurrent Neural Networks are mighty since they combine two properties: a distributed hidden state that allows them to store a large amount of knowledge about the past effectively and non-linear dynamics that permit them to renovate their hidden state in complex ways. Recurrent neural networks can calculate everything that your computer can compute with enough neurons and time.

4. Long-short Term Memory Network

In 1997, Hochreiter and Schmidhuber developed a long-short term memory network to tackle the challenge of getting a recurrent neural network to remember items for a longer time. They created a memory cell with multiplicative interactions utilizing logistic and linear units. When the cell's "write" gate is turned on, data enters the cell. As long as the cell's "keep" gate is turned on, the information remains in the cell. By turning on the cell's "read" gate, information may be retrieved. In 2009, Graves and Schmidhuber demonstrated that recurrent neural networks with long-short term memory are the most effective systems for interpreting cursive writing. In short, instead of using pen coordinates, they employed a series of tiny pictures as input.

5. Boltzmann Machine Network

A stochastic recurrent neural network with a Boltzmann machine is a form of stochastic recurrent neural network. It may be thought of as Hopfield nets' stochastic, generative counterpart. It was one of the first neural networks to learn internal representations and represent and solve complex combinatorial problems.

The Boltzmann machine learning algorithm aims to maximize the probabilities assigned to the binary vectors in training set by the Boltzmann machine. This is the same as maximizing the sum of the log probability assigned to the training vectors by the Boltzmann machine.

6. Hopfield Networks

Non-linear recurrent networks are typically challenging to decipher. These networks can take various forms, such as settling into a stable state, oscillating, or following chaotic paths that are impossible to predict long into the future. A

Hopfield net is made up of binary threshold units connected by recurrent connections. John Hopfield discovered in 1982 that there is a global energy function if the connections are symmetric. The binary threshold decision rule leads the network to settle at least this energy function for each binary "configuration" of the whole network. Hopfield networks have a critical computational function because, rather than storing memories, they are used to generate sensory input interpretations. The visible units represent the input, the states of the hidden units reflect the interpretation, and the energy represents the depravity of the interpretation.

7. Deep Autoencoders

Due to various factors, deep auto-encoders have always appeared to be a pretty good approach to accomplish non-linear dimensionality reduction: They offer flexible mappings in both directions. In terms of the number of training examples, the learning time is linear. The final encoding model is also relatively small and quick. However, employing backpropagation to improve deep autoencoders proved to be quite challenging. The backpropagated gradient dies when the initial weights are modest. We now have far better methods for optimizing them, such as using unsupervised layer-by-layer pre-training or properly initializing the weights in Echo-State Nets.

8. Deep Belief Network

A belief net is a stochastic variable-based directed acyclic network. We can see some of the variables using a belief net. We want to address two problems: the inference issue (infer the states of unobserved variables) and the learning problem (modify the interactions between variables to make the network more likely to create training data). Experts were employed to determine the graph structure and conditional probabilities in early graphical models. Because the networks were poorly linked, researchers initially concentrated on accurate inference rather than learning. Learning was fundamental to neural nets, and hand-writing knowledge was frowned upon because knowledge was derived through learning the training data. To make inference easier, neural networks did not seek interpretability or sparse connectivity.

6.1.4. Advantage and Disadvantages of Deep Learning

While discussing the benefits and challenges of deep learning, we may lack balance in the weighing machine [17]. The benefit side will always be on the heavier side. However, there are a few challenges too, which need not be ignored.

6.1.4.1. Advantages

- Features are automatically inferred and adjusted to get the desired result. It is not necessary to extract features ahead of time. This eliminates the need for time-consuming machine learning approaches.
- The ability to adapt to natural changes in the data is learned automatically.
- Many different applications and data sets can benefit from the same neural network-based technique.
- GPUs can conduct massive parallel computations and are scalable for enormous amounts of data. Furthermore, it provides superior performance outcomes while dealing with large amounts of data.
- The deep learning architecture is adaptable, which means it may solve new issues in the future.

6.1.4.2. Disadvantages

- As data models are complicated data models, training is highly costly. Deep learning also necessitates the use of expensive GPUs and hundreds of computers. The users' costs will rise as a result of this.
- To perform better than other approaches, it needs a vast volume of data.
- It is challenging to interpret output based only on learning and thus necessitates the use of classifiers. Such tasks are carried out using algorithms based on convolutional neural networks.
- Since it necessitates topology, training technique, and other factors, there is no standard theory to help you choose the proper deep learning tools. As a result, it is difficult for less competent individuals to support it.

6.1.5. Application of Deep Learning

As we have learned so far, deep learning seems to have an innumerable variety of applications to perform. Amid all, a few of the most notable applications are given below:

1. Automatic driving automobiles
2. Automatic game playing
3. Automatic machine translation
4. Colorization of black and white images
5. Adding sounds to silent movies
6. Automatic handwriting production
7. Image caption generation
8. Character text generation
9. Mitosis identification from huge pictures
10. Toxicity detection for various chemical compounds
11. Automated essay scoring program for assessing student essays
12. Sequence generation

6.2. ARTIFICIAL NEURAL NETWORK

6.2.1. Overview

Do you ever consider what it's like to construct something as complex as a brain, how these devices function, or what they accomplish? Let's examine how nodes communicate with neurons and how artificial and organic neural networks vary.

Synapse, dendrites, cell body, and axon are parts of a biological neural network (BNN). Neurons do the processing in this neural network. Dendrites receive messages from other neurons, the soma adds up all of the signals, and the axon sends the signals to other cells. The organic nervous system heavily influences artificial neural networks. It is, therefore, quite beneficial to have some understanding of how this system works. Most living things that have the potential to adapt to a changing environment require a learning control unit. To execute tasks,

higher-developed animals and humans rely on very intricate networks of highly specialized neurons.

The control unit, or brain, is split into structural and functional subunits, each with its own set of functions, such as vision, hearing, motor control, and sensor control. Nerves link the brain to the sensors and actors throughout the body. The brain has a vast number of neurons, around 10^{11} on average. These are the fundamental components of the central nervous system (CNS). Synapses are spots on the surface of the neurons that link them. The brain's complexity stems from the vast number of highly linked basic units that function in parallel, with each neuron getting information from up to 10,000 other neurons.

The neuron is made up of all of the structures that make up an animal cell. In a primary cell, the structure and operations are enormously complicated. Even the most advanced neuron models in artificial neural networks appear to be toy-like in comparison. The cell body (soma), dendrites, and axon are the three primary structural components of the neuron. The neuron's organelles are housed in the cell body, and the 'dendrites' originate there. These are slender, widely branched fibers that link to many cells inside the cluster by branching out in diverse directions. Input connections are formed from other cells' axons to the dendrites or directly to the cell's body. Axodentritic and axonsomatic synapses are two types of synapses. Each neuron has only one axon. It's a single, long fiber that carries the cell's output signal as electrical impulses along its length. The axon's end can split into many branches, which are subsequently linked to other cells. The function of the branches is to spread out the signal to a large number of different inputs. Fig. (6.1) illustrates the similarity between the biological neural network and artificial neural network.

The neurons essentially perform the following function: they sum up all of the cell's inputs, which may differ depending on the intensity of the connection or the frequency of the incoming signal.

A threshold function is used to process the input sum and generate an output signal. Compared to a modern computer, the processing time of each cycle is around one milliseconds, while the transmission speed of the neurons is about 0.6 to 120 milliseconds. In roughly 100 milliseconds, a human can identify the image of another individual. Given an individual neuron's processing time of 1 millisecond, this implies that a small number of neurons, less than 100, are involved in serial

processing. However, the task's complexity supports parallel processing, as a problematic recognition task cannot be performed by such a small number of neurons. The 100-step-rule is a term used to describe this occurrence.

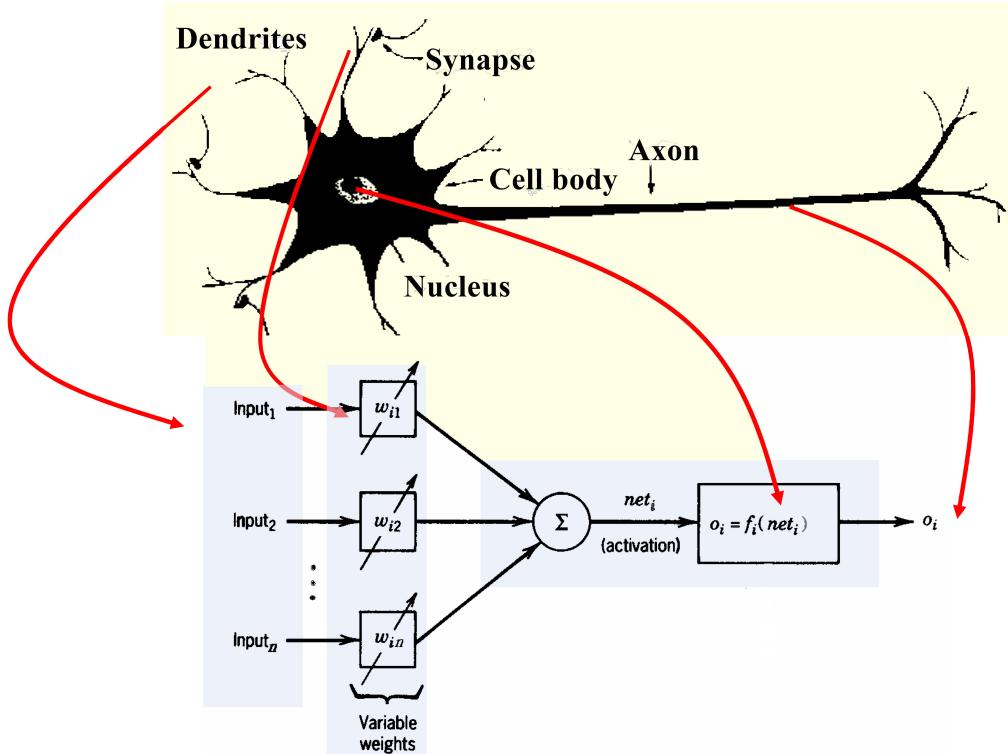


Fig. (6.1). The similarity of biological neural network (BNN) and artificial neural network (ANN).

Artificial Neural Network and Neural Computing phrases have a wide range of definitions and interpretations in the literature. The definitions that follow are geared toward computing, but they are still quite thorough, in my opinion, and they cover a wide range of perspectives on what an ANN is.

Igor Aleksander's concept encompasses a wide variety of methodologies and applications in the field of neural computing:

“Neural computing is the study of networks of adaptable nodes which, through a process of learning from task examples, store experimental knowledge and make it available for use.”

Laurene Fausett of artificial neural networks uses only the connectionist research technique. He said an artificial neural network is a data-processing system with the same performance characteristics as biological neural networks. Artificial neural networks were created as extensions of mathematical models of human cognition or neural biology, based on the following assumptions:

1. Information is processed by a large number of essential components known as neurons.
2. Signals are transferred from one neuron to the next *via* connecting connections.
3. Each connecting link has a weight associated with it, doubting the signal conveyed in a conventional neural network.
4. To calculate its output signal, each neuron applies an activation function to its net input.

A neural network is a computer model with layers of linked nodes that simulates the networked organization of neurons in the brain. It can be trained to identify patterns, categorize data, and predict future occurrences using data [17].

Artificial neural networks are systems that are based on biological neural networks. Without any task-specific rules, these systems learn to execute tasks by being exposed to various datasets and examples. Instead of being programmed with a pre-coded understanding of these datasets, the system identifies features from the given data.

So basically, we can jot down some points for an artificial neural network, which are as follows:

- It comprises linked modules that are basic in design.
- It is built on module-to-module communication.
- Parallel processing is used to complete its purpose.
- It is tolerant of faults.

- It is a method of learning through doing.
- It possesses the capacity to generalize.
- Due to the overall architecture, it is capable of performing complicated tasks.

Memorization and generalization skills are required to imitate intelligent behavior. These are the fundamental characteristics of artificial neural networks. The Collins English Dictionary has the following definitions:

“To memorize: to commit to memory; learn to remember.”

“To generalize: to form general principles or conclusions from detailed facts, experience, etc.”

Given facts, memorizing is an apparent activity in learning. This can be accomplished by either directly saving the input samples or recognizing the idea underlying the input data and learning its general principles. The system's capacity to discover and generalize rules allows it to make predictions based on unknown inputs.

In addition, generalization eliminates the requirement for a large number of input samples to be stored. The system just has to remember which characteristics are part of the sample rather than repeating features common to the whole class for each sample. This can drastically minimize the amount of memory required and result in a very practical memorizing approach.

To comprehend neural networks, we must first dissect and comprehend the most fundamental unit of a Neural Network, the Perceptron [18].

A single-layer neural network called a perceptron is used to categorize linear data.

It is made up of four key elements: (i) Inputs, (ii) Bias and Weights (iii) Function of Summation Function (iv) Transformation or activation Function.

6.2.2. Perceptron Learning

A neural network's basic unit is the perceptron. It is a primary neuron used to categorize its input into one of two categories, initially suggested by Rosenblatt

(1958). Perceptron, like neurons, is a network that receives some inputs, processes them, and provides an output.

A single-layer neural network called a perceptron is used to categorize linear data. The perceptron is a machine learning method for supervised categorization of input into multiple non-binary outputs.

A perceptron is a single artificial neuron that uses a threshold activation function or a step function to compute its weighted input. For binary classification, the Perceptron is employed. Fig. (6.2) shows a perceptron with input x , weights w and bias b , giving output y .

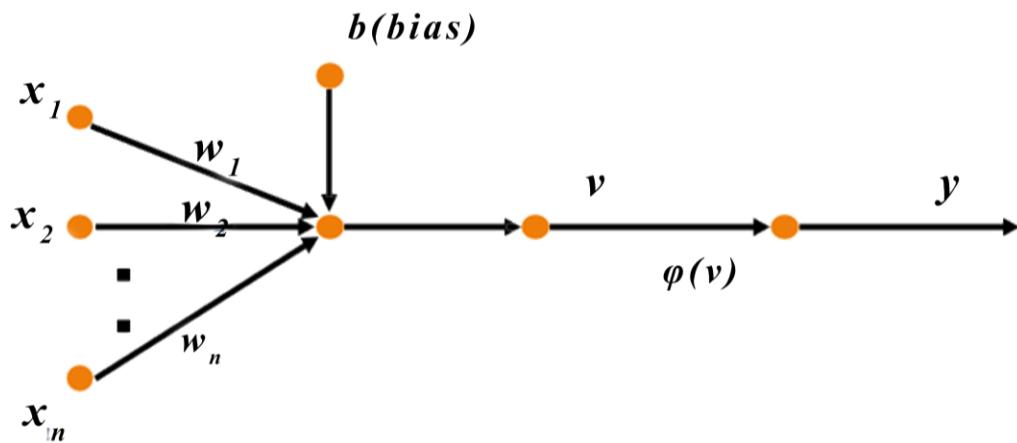


Fig. (6.2). A perceptron with input x , weights w and bias b , giving output y .

Where, $\varphi(v) = +1$ (if $v \geq 0$); otherwise, $\varphi(v) = -1$

A perceptron is a neural network's solitary processing unit. It utilizes a step function that yields +1 if the weighted total of its inputs is greater than 0 and -1 if it is less than 0.

While the dendrite of real neurons receives electrical impulses from the axons of other neurons, these electrical signals are represented numerically in the perceptron. Electrical impulses are modified in varying degrees at synapses between dendrites and axons. In the perceptron, this is also replicated by multiplying each input value by a weight value.

Only when the overall intensity of the input signals exceeds a particular threshold does a real neuron fire an output signal. In a perceptron, we mimic this phenomenon by computing the weighted sum of the inputs, which represents the overall intensity of the input signals, and then applying a step function to the sum to produce the output. This output is transmitted to other perceptrons, much like in biological neural networks.

6.2.2.1. Linear Threshold Unit (TLU)

A perceptron is a single artificial neuron that uses a threshold activation function or a step function to calculate its weighted input. It's also known as a TLU (Threshold Logical Unit), as seen in Fig. (6.3).

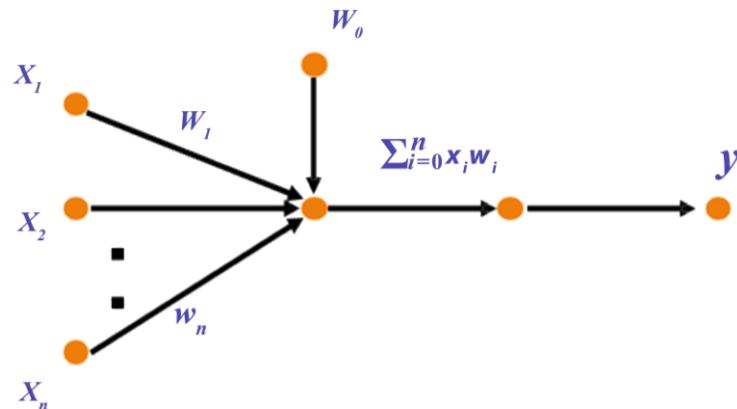


Fig. (6.3). Threshold logical unit (TLU).

$$f(x_i) = +1 \left(\text{if } \sum_{i=0}^n x_i w_i > 0 \right); \text{otherwise } f(x_i) = -1$$

The perceptron is a machine learning method for supervised categorization of input into multiple non-binary outputs.

A perceptron is a single artificial neuron that uses a threshold activation function or a step function to calculate its weighted input.

The Perceptron can only model linearly separable classes.

- First, create a classification job for a perceptron.
- Find appropriate weights so that the training examples are adequately categorized.
- Try to find a hyperplane that divides the instances of the two classes geometrically.

❖ **Linear Separability**

The notion of linear separability states that the input space is divided into areas based on the condition of network response, be it positive or negative.

If two sets of points in 2-D space can be divided by a straight line, they are said to be linearly separable. In general, if a **hyperplane** of $(n-1)$ -dimensions separates two sets of points in n -dimensional space, they are linearly separable.

Assume a network with a positive response in the first quadrant and a negative response in all other quadrants (AND function) using binary or bipolar data; the decision line between the positive and negative response regions is then constructed. As a result, the AND gate is linearly separable, making it simple to build using the perceptron learning technique.

The perceptron can hardly represent linearly separable functions or functions that can be shown in a two-dimensional graph with a single straight line separating two parts of the values. It cannot, however, simulate the XOR function since it is not linearly separable. When the two data points are not linearly separable, a linear separator that minimizes the mean squared error may be desired.

6.2.2.2. Single Layer Perceptron Learning

An input and output layer makes up a Single Layer Perceptron. A complex limiting function is used as the activation function. The Single Layer Perceptron is defined as a configuration of one input layer of neurons feeding forward to one output layer of neurons. The earliest suggested neural model was the single-layer perceptron. A vector of weights makes up the content of the neuron's local memory. A single layer perceptron is computed by calculating the sum of the input vectors, each with the value multiplied by the corresponding member of the weights vector. The value presented in the output will be the activation function's input. Fig. (6.4) illustrates a single layer perceptron network.

$$y_j = f(\text{net}_j) = \begin{cases} 1 & \text{if } \text{net}_j \geq 0 \\ 0 & \text{if } \text{net}_j < 0 \end{cases} \quad \text{where } \text{net}_j = \sum_{i=1}^n x_i w_{ij}$$

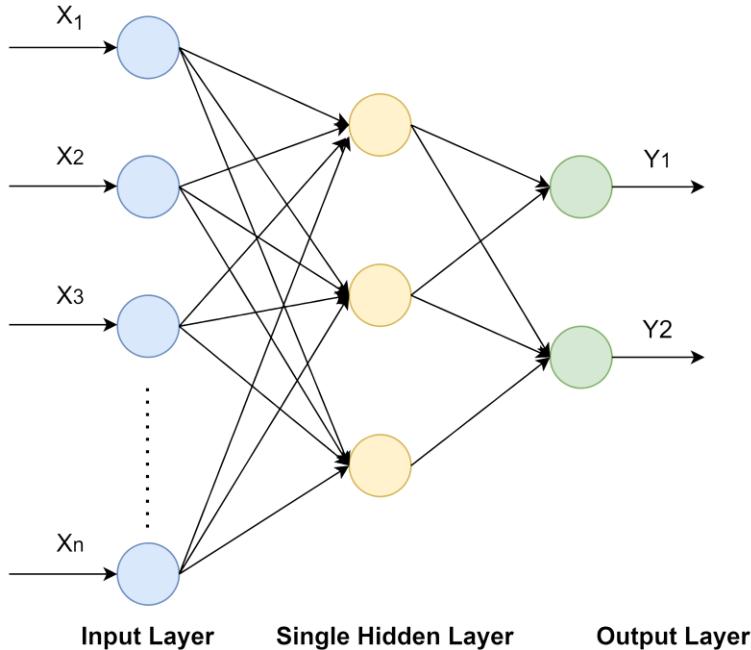


Fig. (6.4). Single layer perceptron network.

❖ Algorithm

Step 1: Create a perceptron with $(n + 1)$ input neurons x_0, x_1, \dots, x_n , using $x_0 = 1$ as the bias input. Assume that O is the output neuron.

Step 2: Set weight $\mathbf{W} = (\mathbf{w}_0, \mathbf{w}_1, \dots, \mathbf{w}_n)$ with some random values.

Step 3: Using the weight set, iterate over the training set's input patterns \mathbf{x}_j , computing the weighted sum of inputs $\text{net}_j = \mathbf{x}_j \mathbf{w}_j$ for $i = 1$ to n for each input pattern j .

Step 4: Using the step function, as given below, compute the result y_j .

$$y_j = f(\text{net}_j) = \begin{cases} 1 & \text{if } \text{net}_j \geq 0 \\ 0 & \text{if } \text{net}_j < 0 \end{cases} \quad \text{where } \text{net}_j = \sum_{i=1}^n x_i w_{ij}$$

Step 5: For each input pattern j , compare the computed output y_j with the intended output y_j . If all of the input patterns were successfully categorized, output the weights and quit the program.

Step 6: If not, adjust the weights as follows:

If the calculated outputs y_j is 1 when it should be 0, then $w_i = w_i - \alpha x_i$, where $i = 0, 1, 2, \dots, n$

If the computed outputs y_j are zero when they should be one, then $w_i = w_i + \alpha x_i$, $i = 0, 1, 2, \dots, n$, where α is a constant value and the learning parameter.

Step 7: Return to step 3

Step 8: Quit the program

6.2.2.3. Multilayer Perceptron Learning

The most common form of neural network in use today is multilayer perceptrons (MLP). They are a form of feedforward neural network, a fundamental type of neural network capable of approximating generic classes of functions, such as continuous and integrable functions.

Perceptrons are building blocks that only become effective in bigger functions like multilayer perceptrons, much as Rosenblatt founded the perceptron on a McCulloch-Pitts neuron, proposed in 1943.

The multilayer perceptron is deep learning's "hello world": a fantastic spot to start learning about deep learning. A multilayer perceptron (MLP) is a type of artificial neural network that has several layers. More than one hidden layers with any number of units make up a multilayer perceptron. In the input layers, linear combination functions are used. In the hidden layers, it employs sigmoid activation functions. It has any number of outputs and may be activated in any way. Any continuous function may be approximated using MLPs with one hidden layer.

In supervised learning applications, multilayer perceptrons are frequently used. They learn to represent the correlation between inputs and outputs by training on a collection of input-output pairings. To decrease error, the model's parameters, or weights and biases, are adjusted throughout training. Backpropagation is used to modify the weight and bias concerning the error, which may be assessed in a range of techniques, including root mean squared error.

MLP, like any other feed-forward neural network, is primarily concerned with two motions: back and forward. Since each guess is a test of what we believe we know, and each response is feedback letting us know how incorrect we are, you might think of this ping pong of guesses and replies as a type of pendulum.

The signal flows from the input layer *via* the hidden layers to the output layer in the forward pass, and the output layer's decision is compared to the ground truth labels. In the backward pass, partial derivatives of the error function w.r.t. the different weights and biases are back-propagated through the MLP using backpropagation and the chain rule of calculus. The differentiation operation creates a gradient, or error landscape, along which the parameters may be tweaked to get the MLP closer to the error minimum. Any gradient-based optimization technique, such as stochastic gradient descent, can be used to do this. Fig. (6.5) shows multi-layer perceptron network.

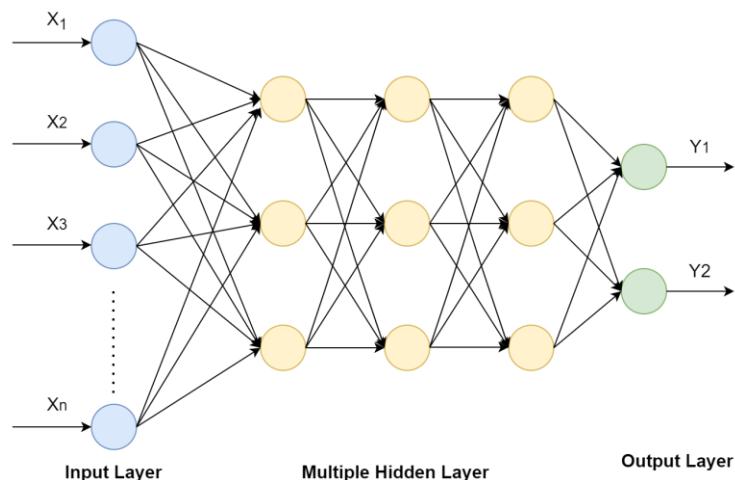


Fig. (6.5). Multi-layer perceptron network (MLP).

6.2.3. Working Principle of Artificial Neural Network

1. Artificial neural networks may be believed as weighted directed graphs with nodes representing artificial neurons and directed edges with weights representing connections between neuron outputs and neuron inputs [19].
2. The artificial neural network gets data from the outside environment in the form of patterns and vector images. The notation $x(n)$ for n number of inputs designates these inputs.
3. Each input is multiplied by the weights assigned to it. The information that the neural network uses to solve a problem is called weights. The strength of the connectivity between neurons inside the neural network is often represented by weight.
4. Inside the computing unit, all of the weighted inputs are added together. If the weighted sum is zero, bias is applied to the output to make it non-zero or scale up the system response. The weight and input of bias are always set to one.
5. Any numerical number between 0 and ∞ corresponds to the sum. The threshold value is chosen to limit the response so that it reaches the desired value. The sum is sent forward through an activation function in this case.
6. To obtain the desired result, the activation function is set to the transfer function. There are two types of activation functions: linear and nonlinear.

6.2.4. Types of Neural Network

Artificial neural networks are of many types. Among them, there are eight most prominent kinds of artificial neural networks [19], which are as follows:

1. Modular Neural Network

Many neural networks work together to produce the outcomes in this kind of neural network. Each of these neural networks performs and constructs a variety of sub-tasks. When compared to other neural networks, this gives a unique collection of inputs. To complete any job, there is no signal exchange or contact between these neural networks. When the number of connections is reduced, the calculation performance improves, reducing the requirement for neural networks to

communicate [18]. The overall processing time will also be determined by the number of neurons, which participate in the computation of outcomes and their engagement in the process. Modular neural networks are one of AI's fastest-growing fields.

2. Single Layer Perceptron

Frank Rosenbluth introduced the single-layer perceptron as the first neural network model in 1958. It is one of the first learning models. The weight vector and the bias parameter are used to calculate a linear decision function. Understanding artificial neural networks are required to understanding the perceptron layer.

3. Multilayer Perceptron

A multilayer perceptron is a form of feedforward neural network. The name MLP is ambiguous; it can apply to any feedforward ANN, or it can refer specifically to networks made up of several layers of perceptrons. Multilayer perceptrons, especially those with a single hidden layer, are frequently referred to as "vanilla" neural networks. There are at least three nodes in an MLP: an input layer, a hidden layer, and an output layer. Except for the input nodes, each node is a neuron with a nonlinear activation function.

4. Feedforward Neural Network

The feedforward neural network is the purest form of an ANN since the information only goes in one way. Data enters through input nodes and exits through output nodes in this type of neural network, including hidden layers. This neural network employs a classifying activation function. Backpropagation is not permitted, and only forward propagation is permitted.

5. Radial basis function Neural Network

A Radial Basis Function (RBF) neural network is a three-layer feedforward neural network. The first layer corresponds to the network's inputs, the second to a hidden layer made up of a number of RBF non-linear activation units, and the third to the network's final output. In RBFNs, activation functions are often implemented as Gaussian functions.

6. Self Organizing Neural Network

Vectors from any dimension are fed into a discrete map in this neural network. The map is used to produce training data for an organization. The map might have one or two dimensions. Depending on the value, the weight of the neurons may fluctuate. Recognizing data patterns is one of the effective uses of the Kohonen Neural Network (also known as SOM network). It is also utilized in medical research to categorize illnesses more accurately. After evaluating the trends in the data, the data is grouped into distinct groups.

7. Hopfield Network

Dr. John J. Hopfield created the Hopfield neural network in 1982. It is made up of one or more completely linked recurrent neurons in a single layer. For auto-association and optimization tasks, the Hopfield network is widely employed. There are two basic types: discrete and continuous. Discrete works in a discrete line way, which means that the input and output patterns are discrete vectors, which can be binary (0,1) or bipolar (+1,1). There are no self-connections in the network, which has symmetrical weights. In the continuous type of network, time is a continuous variable. It is also utilized in auto association and optimization issues like the traveling salesman.

8. Boltzman Machine Neural Network

These networks resemble the Hopfield network, except that some neurons are input and others are concealed in nature. Here the weights are chosen randomly and then learned using the backpropagation method.

6.2.5. Applications of Neural Network

Neural networks have been used effectively in a wide range of data-intensive applications, including:

- Process modeling and control
- Machine Diagnostics
- Target Recognition
- Portfolio Management

- Credit Rating
- Voice recognition
- Financial Forecasting
- Fraud detection

6.2.6. Programming Approach for Artificial Neural Network (ANN)

```
# 1. Import libraries

# Import tensorflow and keras
import tensorflow as tf
from tensorflow import keras

# Import other libraries
import numpy as np
import matplotlib.pyplot as plt

# 2. Import dataset

# Import fashion mnist dataset
dataset = keras.datasets.fashion_mnist

# Split dataset to train and test
(trainX, trainY), (testX, testY) = dataset.load_data()

# Set data labels
data_labels = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
               'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']

trainY.shape

(60000,)

# 3. Export example datas
plt.figure(figsize=(10,10)) # Predict size of print
for i in range(49):
    plt.subplot(7,7,i+1) # Print 7*7 datas
    plt.xticks([]) # Not using x-axis graduation
```

```

plt.yticks([]) # Not using y-axis graduation
plt.grid(False) # Not using grid
plt.imshow(trainX[i], cmap=plt.cm.binary) # Print images
plt.xlabel(data_labels[trainY[i]]) # Print labels
plt.show() # Print conclusion

```



```

# 4. Make ANN model
ANN = keras.Sequential([
    keras.layers.Flatten(input_shape=(28, 28)), # Reduce dimension
    keras.layers.Dense(256, activation='relu'), # Fully connected layer
    keras.layers.Dense(10, activation='softmax') # Fully connected layer with output layer
])
ANN.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
            metrics=['accuracy']) # Compile model
ANN.summary() # Check model

```

Model: "sequential"

Layer (type)	Output Shape	Param #
flatten (Flatten)	(None, 784)	0

dense (Dense)	(None, 256)	200960
dense_1 (Dense)	(None, 10)	2570
<hr/>		
Total params: 203,530		
Trainable params: 203,530		
Non-trainable params: 0		

```

# 5. Learning model
ANN.fit(trainX, trainY, epochs=5)

Epoch 1/5
1875/1875 [=====] - 6s 3ms/step - loss: 3.6962 - accuracy: 0.7367
Epoch 2/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.6083 - accuracy: 0.7863
Epoch 3/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.5426 - accuracy: 0.8107
Epoch 4/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.5054 - accuracy: 0.8242
Epoch 5/5
1875/1875 [=====] - 5s 3ms/step - loss: 0.4857 - accuracy: 0.8314
<tensorflow.python.keras.callbacks.History at 0x7f0b040ad290>

# 6. Evaluate model
loss, acc = ANN.evaluate(testX, testY, verbose=2) # Calculate Accuracy
print('Test accuracy : ', acc) # Print result

313/313 - 0s - loss: 0.5463 - accuracy: 0.8207
Test accuracy : 0.8206999897956848

# 7. Test
result = ANN.predict(testX)

# Print test result
for i in range(len(result)):
    print('number : ', i) # Print number of test
    print('actual : ', testY[i], data_labels[testY[i]]) # Print actual label

```

```
print('predict : ', np.argmax(result[i]), data_labels[np.argmax(result[i])]) # Print predicted label
print('\n')

number : 9000
actual : 6 Shirt
predict : 6 Shirt

number : 9001
actual : 9 Ankle boot
predict : 9 Ankle boot

number : 9002
actual : 0 T-shirt/top
predict : 3 Dress

number : 9003
actual : 1 Trouser
predict : 1 Trouser

number : 9004
actual : 6 Shirt
predict : 6 Shirt
```

6.3. COMPONENTS OF NEURAL NETWORK

6.3.1. Layers

In the past section, we have discussed artificial neural networks. This section deals with all the components and functions of a neural network. A neural network decomposes your information into layers of abstraction. An input layer, hidden layers, and an output layer make up the structure. The layers are linked together *via* nodes, or neurons, with each layer using the preceding layer's output as its input. Receiving a set of inputs, performing computations, and then using the result to solve the problem is the primary function.

The main three components of a neural network are the layers, input, output, and hidden layers.

6.3.1.1. Input Layer

A layer is a collection of neurons that are grouped. It is utilized to keep a group of neurons together. Within each layer of a neural network, the neurons execute the same function. They simply add the bias and perform an activation function after calculating the weighted sum of inputs and weights. Fig. (6.6) shows the typical input layer.

Artificial input neurons comprise up the input layer of a neural network, which delivers the initial data into the system for processing by succeeding layers of artificial neurons. The input layer is the first step in the artificial neural network's workflow. Since the input layer is the initial layer of the network, artificial neurons in the input layer have a distinctive function to perform. Experts explain this by the input layer being made up of passive (not performing any operations) neurons that do not take in data from preceding levels.

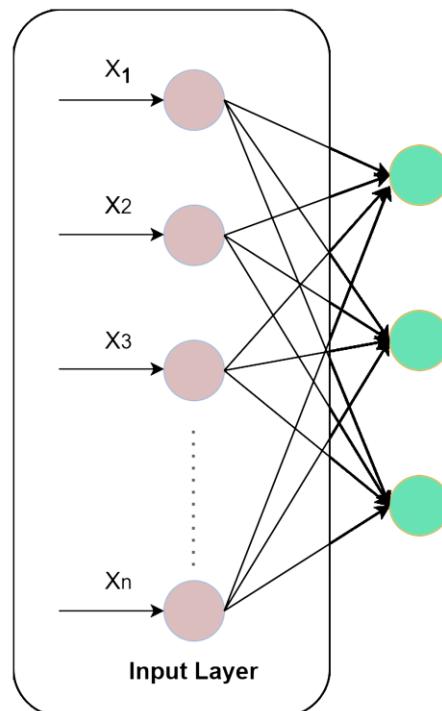


Fig. (6.6). Input layer.

In particular, artificial neurons have a set of weighted inputs and functions based on those weighted inputs. However, an input layer can theoretically be made up of artificial neurons with no weighted inputs or calculated differently. For example, randomly, because the information is entering the system for the first time. The data is sent from the input layer to the succeeding layers in the neural network model, where the neurons have weighted inputs.

The input layer is in charge of receiving the data. These inputs can be retrieved from a remote location. In a neural network, there must always be one input layer. The input layer receives the data, conducts the computations using its neurons, and then sends the results to the next layers.

6.3.1.2. Output Layer

The output layer of an artificial neural network is the final layer of neurons that generates the program's outputs. Given that output layer neurons are the last "actor" nodes on the network, they may be constructed or monitored differently from other artificial neurons in the neural network. Fig. (6.7) shows the output layer.

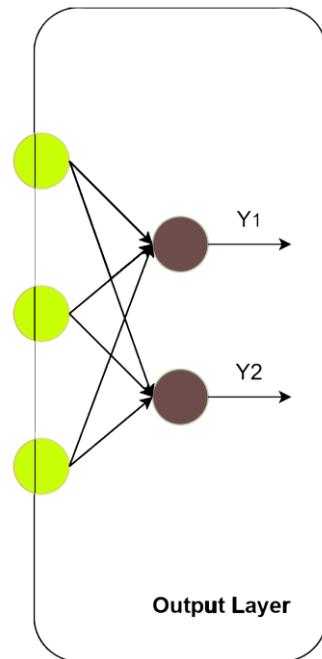


Fig. (6.7). Output layer.

The output layer is in charge of generating the final product. In a neural network, there must always be one output layer. The output layer receives the inputs from the layers above it, conducts the computations using its neurons, and then computes the output.

The output layer, in a way, condenses and concretely generates the ultimate product. However, to fully comprehend the neural network, it is necessary to examine the input layer, hidden layers, and output layer as a whole.

6.3.1.3. *Hidden Layer*

Neural networks now outperform most machine learning methods because of the addition of hidden layers. Hidden layers are located between the input and output layers. They are not visible to external systems and are private to the neural network, as the name suggests. In a neural network, there might be one or more hidden layers. Fig. (6.8) shows the hidden layer of a neural network.

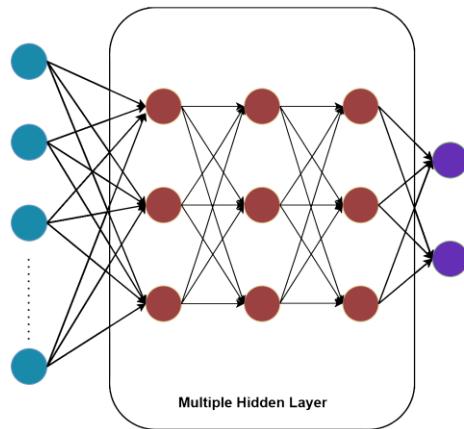


Fig. (6.8). Hidden layers.

The more hidden layers in a neural network, the longer it will take for the network to create an output and the more complicated issues it will handle. The neurons add the bias and determine the weighted sum of inputs and weights before performing an activation function. Hidden layers enable a neural network's function to be split down into particular data manipulations. Each function in the hidden layer is tailored to deliver an inevitable result. For example, hidden layer algorithms that recognize human eyes and ears may be used with later layers to identify faces in

pictures. While the functions for identifying eyes on their own are insufficient for independently recognizing objects, they can operate together in a neural network.

The design of hidden layers in the neural network is the subject of several machine learning model evaluations. There are various ways to build up these hidden layers for creating diverse outcomes, such as convolutional neural networks for image processing, recurrent neural networks with memory, and basic feedforward neural networks that straightforwardly function on training data sets.

6.3.2. Weights and Bias of a Neuron

This section attempts to give fundamental insight into bias and weights. The essential notion in a neural network is its weights and bias. The weights are multiplied with the inputs and fed into an activation function. The bias is added when the inputs are transferred across neurons.

6.3.2.1. Weights

The coefficients of the equation are called weights. Negative weights lower the output's value. When a neural network is trained on a training set, it is given a set of weights, to begin with. The optimal weights are then created by optimizing these weights during the training phase.

Weight represents the strength of the link between units. If the weight from node 1 to node 2 is high, neuron 1 has a more substantial impact on neuron 2. The relevance of the input value is reduced when it is given a weight. If the weights are close to zero, altering this input will have little effect on the output. Negative weights indicate that increasing this input will result in a decrease in output. Weight determines how much of an impact the input has on the output. The weighted sum of the inputs is computed initially by a neuron.

To understand how weights operate, it's helpful to picture a theoretical neural network. An input layer of a neural network receives input signals and sends them on to the next layer. The neural network then has a succession of hidden layers that apply modifications to the data input. The weights are added to the nodes of the hidden layers' nodes. For example, before transferring the data to the next layer, a single node may multiply the input data by a specified weight value, then add a bias.

The weighted sum of the inputs is computed initially by a neuron.

$$y = \sum_{i=1}^n w_i * x_i + b$$

For example, if the inputs are:

$$X = (x_1, x_2, x_3, \dots, x_n)$$

The following are the weights:

$$W = (w_1, w_2, w_3, \dots, w_n)$$

After that, a weighted total is calculated as follows:

$$\text{Weighted_Sum} = (w_1x_1 + w_2x_2 + w_3x_3 + w_nx_n)$$

After that, the weighted sum is given a bias.

$$\text{Weighted_Sum} = (w_1x_1 + w_2x_2 + w_3x_3 + w_nx_n) + \text{bias}$$

Finally, the computed value is sent to the activation function, which generates a result.

$$\text{Output for next layer} = \sum_{i=1}^n (w_1x_1 + w_2x_2 + w_3x_3 + w_nx_n) + b$$

6.3.2.2. Bias

Adding a constant value to the product of inputs and weights is known as bias. To compensate for the outcome, bias is used. The bias is used to move the activation function's outcome to the positive or negative side. It is an additional input to neurons with a constant value of 1 and its connection weight. This ensures that the neuron will be activated even if all of the inputs are zero.

Assume your neural network is supposed to output 10 when the input is 5 and the weight is 1. How will you assure that the network's neuron returns 5 since the total weight and input will equal 5? The answer is that a bias of 5 can be added.

For example, if $w = 1; x = 5$ and $y_{expected} = 10$;

$$\text{Then take } b = 5. \text{ (as we know, } y = \sum_{i=1}^n w_i * x_i + b)$$

Since bias is the inverse of the threshold, it determines when the activation function is activated. The inclusion of bias decreases variance, which gives the neural network more flexibility and generalization.

❖ Bias vs. Weight

Inside the network, both weights and bias are learnable parameters. Before learning begins, a teachable neural network will randomize both the weight and bias parameters. Both parameters are changed as training progresses toward the desired values and the correct output. The amount to which the two factors impact the input data differs. Simply said, bias is the distance between the predicted value and the desired value. Biases make up the discrepancy between the function's output and its intended output.

A low bias value indicates that the network makes more assumptions about the output's form. In contrast, a high bias value indicates that the network make fewer assumptions about the output's form. Weights, on the other hand, are a measure of the connection's strength. The degree of effect a change in the input has on the output is determined by its weight. A low weight value will have little impact on the input, whereas a higher weight value will significantly impact the output.

6.3.3. Activation Functions

A mathematical function that takes in an input and creates an output is known as an activation function. When the computed result passes the given threshold, the function is activated. The output of a node is defined by the activation function of that node, given input or collection of inputs. The activation function in a biological network is generally an abstraction reflecting the cell's rate of action potential firing. Non-linearity is introduced to neural networks *via* activation functions. It condenses the values into a narrower range. To encode complicated data patterns,

these functions should be nonlinear. Multi-state activation, Sigmoid, ReLu, etc., are the activation functions utilized in deep neural networks [19].

Consider the activation function to be a mathematical process that normalizes the input and generates a result. The output is subsequently sent to the neurons in the following layer.

Thresholds in the activation function are pre-defined numerical values in the function. The type of activation functions can add the non-linearity of the output. As a result of this characteristic of the activation function, neural networks may handle non-linear problems. Non-linear issues are ones in which the input and output do not have a direct linear connection.

The activation function is also regarded as Transfer Function. It may also be used to connect two neural networks. For an artificial neural network to learn and interpret complicated patterns, activation functions are critical. Its primary purpose is to bring non-linear characteristics into the network. It calculates the ‘weighted sum’, adds a sign, and then chooses whether or not to fire a particular neuron. Their primary function is to transform an ANN node’s input signal into an output signal. That output signal is now used as an input in the network’s following layers. The non-linear activation function will aid the model in comprehending the complexity and providing precise findings.

The output signal would be a simple linear function if we didn’t use an activation function. A linear function is just a one-degree polynomial. Although a linear equation is simple to solve, it is restricted in complexity and has less capacity to learn complicated functional mappings from data. A neural network without an activation function is just a linear regression model, which has limited power and frequently fails to perform well. The neural network would also be unable to learn and represent other complex data types such as pictures, videos, audio, and speech without the activation function.

The activation functions may be split into two categories – linear activation function and non-linear activation functions.

6.3.3.1. Linear Activation Functions

The function is a line or linear, as you can see. As a result, the functions’ output will be unconstrained by any range.

$$f(x) = x ; \text{ Range} = (-\infty \text{ to } +\infty)$$

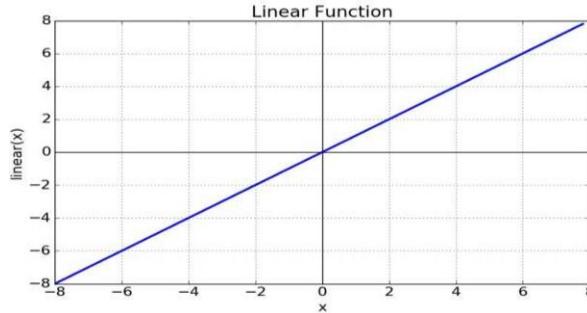


Fig. (6.9). Linear activation function.

It doesn't assist with the complexities of varied characteristics of the typical data supplied to neural networks. Fig. (6.9) depicts the linear activation function's curve.

This section will explain various activation functions widely used in deep learning applications, along with their working principle and nature of the output.

6.3.3.2. Non-linear Activation Functions

When we plot a non-linear function, we see that it has a degree more than one and has a curvature. We now require a neural network model to learn and represent practically any arbitrarily complicated function that links inputs to outputs. As a result, we may construct non-linear mappings from inputs to outputs by employing a non-linear activation function. Fig. (6.10) shows the non-linear activation function.

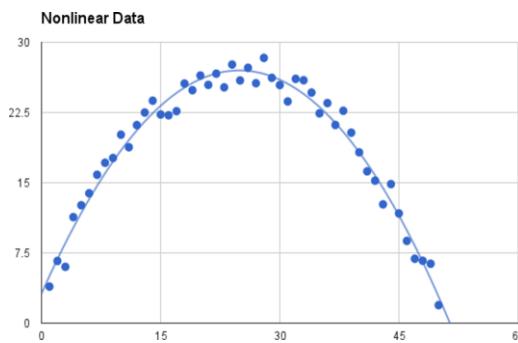


Fig. (6.10). Non-linear activation function.

It allows the model to generalize or adapt to a wide range of data while still distinguishing between the outputs.

6.3.3.3. Multi-state Activation Functions

The multi-state functions use the 2-state Logistic function to do further categorization. The multi-state activation functions show that these activation functions may change the parameter distribution of deep neural networks (DNN) models, improve model performance, and reduce model size. A logistic function, which is monotonically rising and nonlinear with a defining domain $(-\infty, +\infty)$ and a range $(0, 1)$, is commonly employed as an activation function in DNNs. Fig. (6.11) shows the multi-state activation functions. We can frame the equations as:

$$s(x) = \frac{1}{1 + e^{-x}}$$

When you combine several logistic functions, you obtain a multi-logistic function.

$$m(x) = \sum_{i=1}^n s(p_i(x - x_i))$$

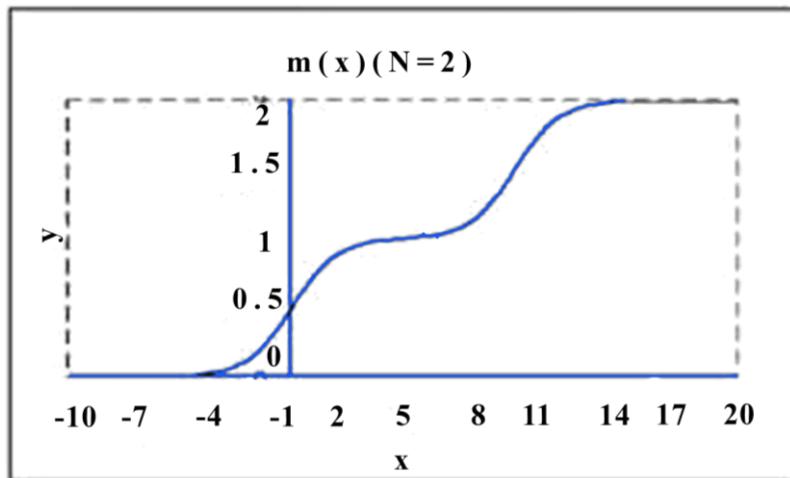


Fig. (6.11). Multistate activation function.

6.3.3.4. Identity Activation Functions

The identity activation function merely adjusts an input by a factor, suggesting that the inputs and outputs have a linear connection. The identity function returns the same value as the input it was given. It's also known as an identity map, identity connection, or identity transformation. If f is a function, then the identity relation for argument x is $f(x) = x$ for all x values. Fig. (6.12) shows the identity activation function.

The mathematical formula is as follows:

$$f(x) = x; \forall x \in R$$

The input is x , and y is a scalar number, such as 2.

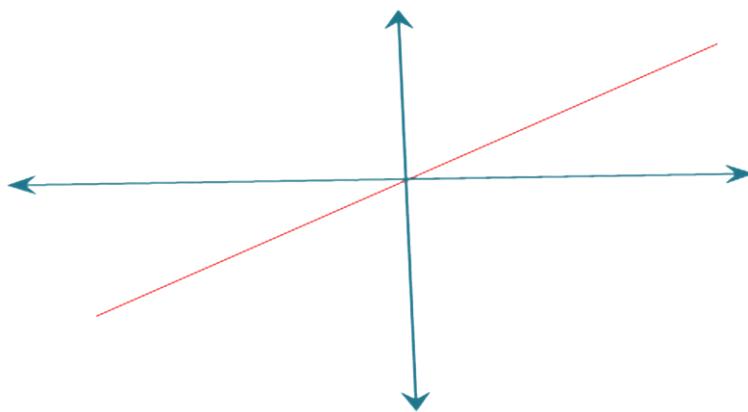


Fig. (6.12). Identity activation function.

6.3.3.5. Binary Step Activation Functions

The Heaviside step function, also known as the unit step function, is a step function named after Oliver Heaviside (1850–1925), whose value is zero for negative inputs and positive arguments. It is typically represented as H or (but sometimes u , 1, or 1). It is an example of the broad category of step functions, which may all be written as linear combinations of this one's translations. It is a linear activation function (Fig. 6.13).

$$f(x) = \begin{cases} 0 & \text{for } x < 0 \\ 1 & \text{for } x \geq 0 \end{cases}$$

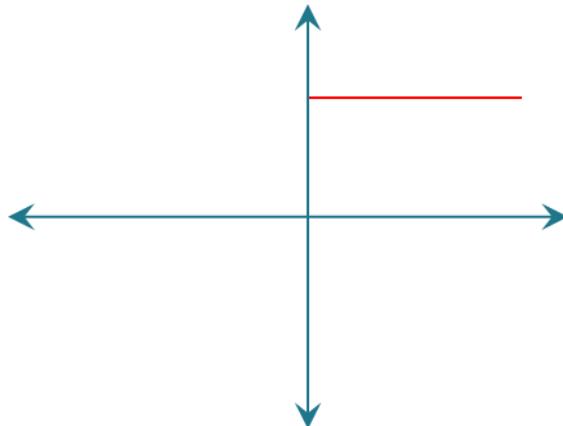


Fig. (6.13). Binary step activation function.

6.3.3.6. Sigmoid Activation Function

The sigmoid or logistic activation function is an S-shaped function curve. We utilize the sigmoid function since it exists between two points (0 to 1). As a result, it is instrumental in models where the probability must be predicted as an output. Because the likelihood of anything only occurs between 0 and 1, sigmoid is the best option.

Sometimes, the logistic sigmoid function can cause a neural network to become stuck during training. However, the beauty of an exponent is that its value in the above equation never approaches zero or exceeds one. The big negative numbers are scaled to 0, while the significant positive values are scaled to 1. Fig. (6.14) is the sigmoid activation function's curve.

The equation for the Sigmoid function is:

$$\begin{aligned} f(x) &= \frac{1}{1 + e^{-x}} \\ f'(x) &= f(x)[1 - f(x)] \end{aligned}$$

Doing the derivative, we get:

$$\begin{aligned}
 \frac{d}{dx} \sigma(s) &= \frac{d}{dx} \left(\frac{1}{1 + e^{-x}} \right) \\
 &= \frac{d}{dx} (1 + e^{-x})^{-1} \\
 &= -(1 + e^{-x})^{-2} (-e^{-x}) \\
 &= \frac{e^{-x}}{(1 + e^{-x})^2}
 \end{aligned}$$

It normalizes an input real value into the range between 0 and 1.

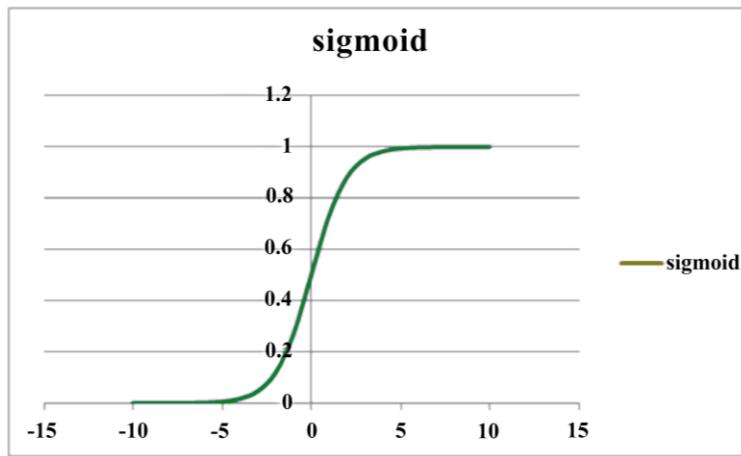


Fig. (6.14). Sigmoid activation function.

6.3.3.7. Tanh Activation Function

Tanh is a sigmoid activation function extension. As a result, Tanh can be utilized to make the output non-linear. The value of the output is between -1 and 1. The Tanh function shifts the sigmoid activation function's outcome. Fig. (6.15) depicts the TanH activation function.

$$f(x) = \text{Tanh}(x) = \frac{2}{1 - e^{-2x}} - 1$$

Because its range is between -1 and 1, its output is now zero centered, *i.e.*, output lies between -1 and +1.

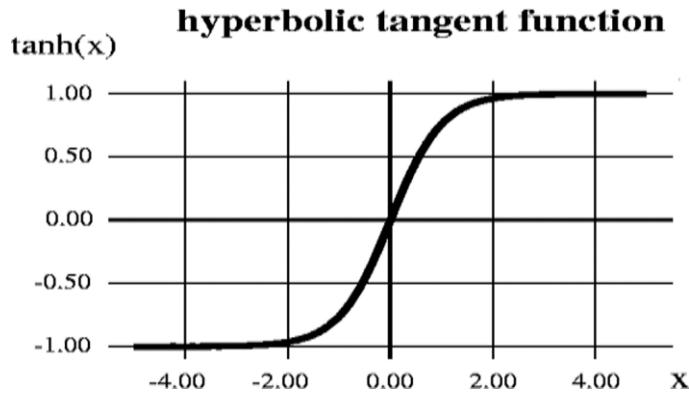


Fig. (6.15). TanH activation function.

As a result, optimization is easy using this approach, and it is always favored over the Sigmoid function in practice. However, it still has the Vanishing gradient issue.

6.3.3.8. Rectified Linear Unit (ReLU) Activation Function

One of the most commonly utilized activation functions is Rectified Linear Unit (ReLU). ReLU should be used in the hidden layer whenever possible. The idea is straightforward. It also gives the output a touch of nonlinearity. The outcome, on the other hand, might range from 0 to infinity.

It was recently demonstrated that it improved convergence by six times over the Tanh function. The positive component of the argument of the ReLU activation function is defined as follows:

$$\begin{aligned}
 R(x) &= x+ = \max(0, x); \text{ where } x \text{ is the input} \\
 &\text{i.e., if } x < 0, R(x) = 0 \text{ and} \\
 &\quad \text{if } x \geq 0, R(x) = x
 \end{aligned}$$

It is also known as the ramp function (Fig. 6.16). Here, in ReLU, the vanishing gradient problem is avoided and corrected.

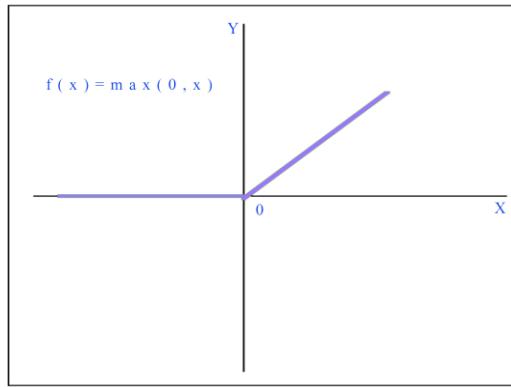


Fig. (6.16). RELU activation function.

However, it should only be utilized within the Hidden layers of a neural network model due to its restriction. Another issue with ReLu is that some gradients are weak and can die during training. It may result in a weight update, after which it will no longer trigger on any data point. Simply said, ReLu has the potential to cause Dead Neurons.

6.3.3.8.1. Advantages of RELU

- Sparse activation: In a network that has been randomly started, only approximately half of the hidden units are active.
- Gradient propagation is improved: When compared to sigmoidal activation functions that saturate in both directions, there are fewer vanishing gradient issues.
- Only comparison, addition, and multiplication are needed for efficient computation.

6.3.3.8.2. Disadvantages of RELU

- It is non-differentiable at zero but differentiable everywhere else, and the derivative value at zero can be arbitrarily selected to be 0 or 1.
- It's not centered at zero.
- Unbounded.

- Dying RELU Problem: ReLU neurons can be driven into states where they are virtually dormant for all inputs at times. Because no gradients travel backward through the neuron in this state, the neuron remains permanently inactive and "dies." This is a variant of the issue of disappearing gradients. Vast quantities of neurons in a network can become trapped in dead states in some situations, thereby reducing model capacity. When the learning rate is set extremely high, this issue occurs. It may be addressed by switching to leaky ReLUs, which assign a slightly positive slope, but performance suffers as a result.

6.3.3.9. Softmax Activation Function

Softmax is a mathematical function that transforms a vector of integers into a vector of probabilities, with the probability of each value proportional to the vector's relative scale. Softmax is an upgraded version of the Sigmoid activation function. The Softmax function adds non-linearity to the output, although it is primarily utilized for classification instances with several classes of outcomes.

The softmax function is most commonly used as an activation function in a neural network model in applied machine learning. The network is set up to produce N values, one for each classification task class, and it is used to normalize the outputs, transforming them from weighted sum values to probabilities that total to one. Each element in the softmax function's output is interpreted as the likelihood of belonging to each class.

$$y = \frac{e^x}{\text{sum}(e^x)}$$

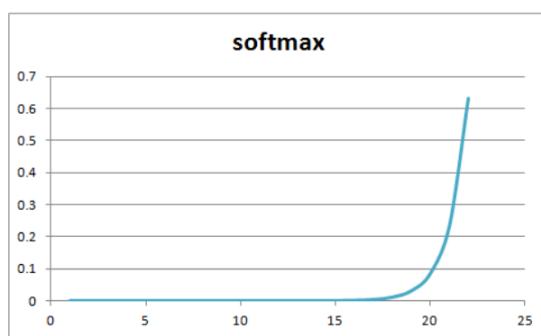


Fig. (6.17). Softmax activation function.

The softmax mathematical function, sometimes known as "softmax," is a probabilistic or "softer" variant of the ArgMax function. Class membership is needed on more than two class labels. It is utilized as the activation function for multi-class classification issues. Fig. (6.17) shows the curve for softmax activation function.

The softmax activation will output one value for each node in the output layer by definition. The output numbers will be probabilities, and the values will total to 1.0. The data must be prepared before modeling a multi-class classification issue. The class labels in the target variable are the first label encoded, which means that each class label is assigned an integer from 0 to N-1, where N is the number of class labels.

The target variables that have been label encoded are then one-hot encoded. This, like the softmax output, is a probabilistic representation of the class label. Each class label and its location are given a position in a vector. All values are set to 0, and a 1 is used to indicate the class label's location.

Three class labels, for example, will be integer encoded as 1, 2, and 3. Then vectors were encoded as follows:

Class 1: [1, 0, 0]

Class 2: [0, 1, 0]

Class 3: [0, 0, 1]

This is stated as “One-hot encoding”. Under supervised learning, it reflects the predicted multinomial probability distribution for each class used to correct the model.

Let's pretend you're working on a neural network that will forecast the likelihood of rain in the future. The softmax activation function, which can compute the likelihood of an event occurring in the future, can be employed in the output layer.

6.3.3.10. Softplus Activation Function

Softplus is a more recent addition to the sigmoid and Tanh functions. It was initially released in 2001. Since Softplus is differentiable and its derivative is

straightforward to explain, it is an excellent alternative to conventional functions. It also has an unexpected derivative, which we will observe here.

The Sigmoid and Tanh functions create outputs with upper and lower bounds, but the Softplus function produces outputs on a scale of $(0, +\infty)$. That is the key distinction.

$$f(x) = \log(1 + e^x)$$

Thus, derivative of $f(x) = f'(x) = \frac{1}{1 + e^{-x}}$

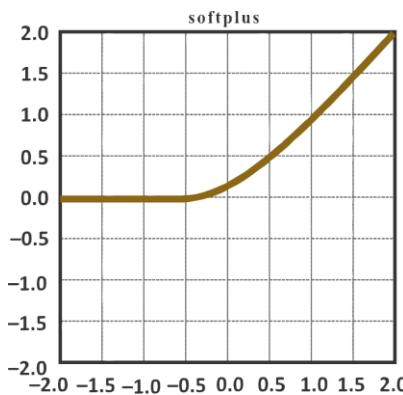


Fig. (6.18). Softplus activation function.

So that's the softplus function's derivative in its simplest form (Fig. 6.18). The derivative is equivalent to the sigmoid function, as you may have seen. Softplus and sigmoid have a similar appearance.

6.3.3.11. Exponential Linear Unit Activation Function

The ELU (Exponential Linear Unit) is a function that tends to converge to zero faster and give more accurate results. Unlike other activation functions, ELU contains an additional alpha constant that must be positive.

With the exception of negative inputs, ELU is quite similar to RELU. For non-negative inputs, they are both in identity function form. ELU, on the other hand, smooths out gradually until its output equals α , whereas RELU smooths out abruptly.

$$f(a, x) = \begin{cases} \alpha(e^x - 1) & \text{for } x < 0 \\ x & \text{for } x \geq 0 \end{cases}$$

Having a range $(-\infty, +\infty)$

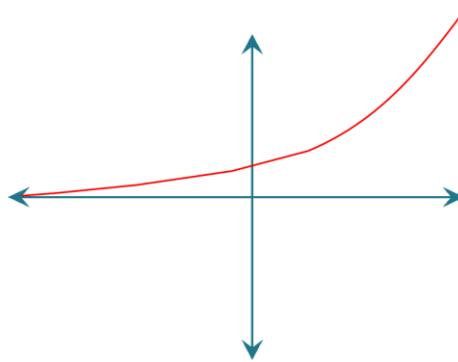


Fig. (6.19). Exponential linear unit activation function.

The exponential linear unit (ELU) is a deep neural network that accelerates learning and improves classification accuracy. Fig. (6.19) shows the curve for exponential linear unit activation function.

Other than these essential activation functions, as listed above, there are few more activation functions are available, which as follows:

1. Arch Tan.
2. Leaky Rectified Linear Unit (LReLU's).
3. Parametric rectified linear unit (PReLU).
4. Randomized leaky rectified linear unit (RReLU).
5. S-shaped rectified linear activation unit (SReLU).
6. Adaptive piecewise linear (APL).
7. SoftExponential.

6.3.4. Forward Propagation

The computation and storing of intermediate variables for a neural network from the input layer to the output layer is called forward propagation [20]. We'll now go through the mechanics of a neural network with one hidden layer one by one. It is

the process of feeding input values to a neural network and receiving a predicted value as an output. Forward propagation is also known as inference. When we feed the input data to the first layer of the neural network, nothing happens. The second layer receives the data from the first layer and multiplies, adds, and activates them before passing them on to the next layer. The procedure is repeated for succeeding levels, and the final layer produces an output value.

6.3.5. Backpropagation

The process of determining the gradient of neural network parameters is known as backpropagation. In a nutshell, the approach follows the chain rule from calculus to traverse the neural network in the opposite order, from the output to the input layer. While calculating the gradient concerning particular parameters, the method saves any intermediate variables that are necessary. Assume we have the functions $\mathbf{y} = \mathbf{f}(\mathbf{x})$ and $\mathbf{z} = \mathbf{g}(\mathbf{y})$, with $\mathbf{x}, \mathbf{y}, \mathbf{z}$ as the input and output tensors of arbitrary forms. We may compute the derivative of \mathbf{z} with respect to \mathbf{x} using the chain rule:

$$\frac{\partial \mathbf{z}}{\partial \mathbf{x}} = \text{product}\left(\frac{\partial \mathbf{z}}{\partial \mathbf{y}}, \frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)$$

This is simple for vectors: it is simply matrix multiplication. We utilize the proper counterpart for higher-dimensional tensors.

We receive an output value, which is the expected value, after forward propagation. We compare the predicted value to the actual output value to compute the error. To compute the error value, we utilize a loss function. The derivative of the error value is then calculated for each and every weight in the neural network. The chain rule of differential calculus is used in backpropagation. The derivatives of the error value with respect to the weight values of the final layer are calculated first in the chain rule. These derivatives are referred to as gradients, and the gradient values are used to compute the gradients of the second final layer.

6.3.6. Learning Rate

The learning rate is a hyperparameter that regulates how much the model changes each time the model weights are altered in response to the predicted error. A too little value may result in a prolonged training process that becomes stuck, whereas

a value too big may result in learning a sub-optimal set of weights too quickly or an unstable training process.

When designing your neural network, the learning rate may be the essential hyperparameter. As a result, it is critical to understand how to explore the impacts of the learning rate on model performance and understand the learning rate's dynamics on model behavior.

To maximize the weights of neural networks, we generally utilize gradient descent. Back-propagation is used to calculate the derivative of the loss function with respect to each weight and remove it from that weight at each iteration. The learning rate defines how fast or slowly you want your parameter values to change. The learning rate should be high enough to ensure that it does not take an infinite time to converge but low enough time to ensure that it discovers the local minima.

6.3.7. Gradient Descent

Gradient descent is one of the most widely used optimization methods, and it is by far the most frequent method for optimizing neural networks. At the same time, every current deep learning package includes implementations of multiple gradient descent optimization techniques.

Gradient descent is a technique for minimizing an objective function $J(\boldsymbol{\theta})$ parameterized by a model's parameters $\boldsymbol{\theta} \in \mathbb{R}^d$ by updating the parameters in the opposite direction of the objective function's gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})$ with respect to the parameters [21]. The magnitude of the steps we take to attain a minimum is determined by the learning rate α . To put it another way, we descend the slope of the surface generated by the goal function until we reach a valley (Fig. 6.20).

Formally,

$$y = f(x_1, x_2, \dots, x_n)$$

Find x_1, x_2, \dots, x_n that maximizes or minimizes the output 'y'. Usually, a cost or loss function or a profit/likelihood function is utilized for the task.

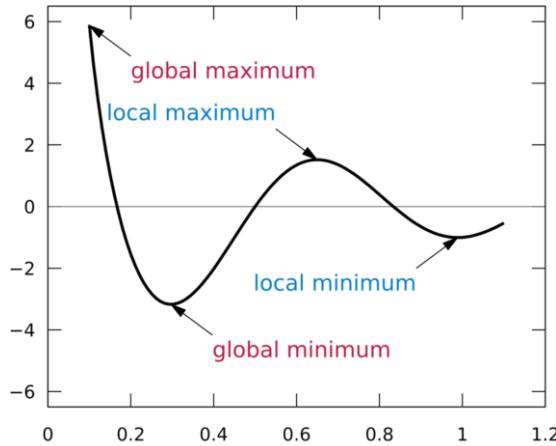


Fig. (6.20). Curve showing the concept of global/local value for maxima/minima.

6.3.7.1. Gradient

The gradient is a single variable which is the derivative or the slope of the tangent line at a point x_0 (Fig. 6.21). A gradient measures the change in all weights in relation to the change in error.

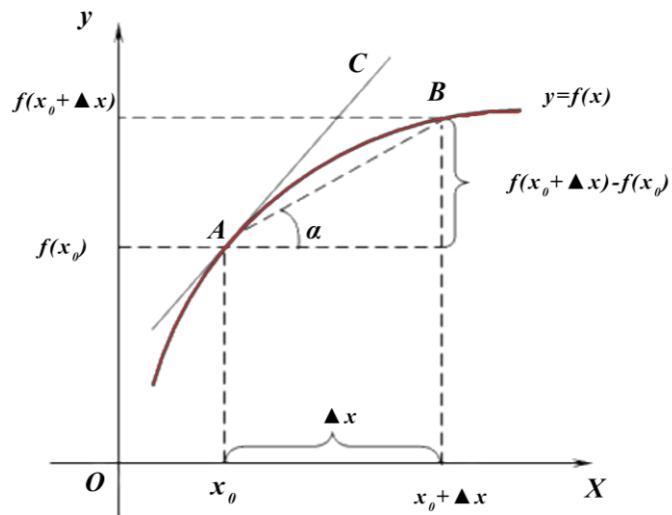


Fig. (6.21). Gradient or slope at a particular point.

The steeper the gradient or slope and the quicker a model can learn, the higher the gradient. If the slope becomes null or zero, the model will cease learning.

6.3.7.2. The General Idea

We have k parameters ($\theta_1, \theta_2, \dots, \theta_k$), we want to train a model to minimize some error or loss function $J(\theta_1, \theta_2, \dots, \theta_k)$. Gradient descent is one method for determining the best set of parameter values iteratively:

1. Set up the settings.
2. Change the values to lower $J(\theta_1, \theta_2, \dots, \theta_k)$.
3. ∇J indicates which direction has the most impact on J .
4. Move-in the opposite direction of ∇J .
5. Keep reducing J by altering the values of $\theta_1, \theta_2, \dots, \theta_k$, until finding a minimum value.

Formally saying:

$$\theta_j = \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} J(\theta_1, \theta_2, \dots, \theta_k)$$

*for $j = 0, 1, \dots, n$ until convergence
where α is the learning rate.*

To make the algorithm sprint, we may select a number that is as high as feasible. While this logic appears correct, there is a risk of overshooting the minima, causing the algorithm to converge slowly or, worse, diverge. As a result, the learning rate α is the step by which the gradient descent algorithm descends in a single iteration.

During a gradient descent, the learning rate α determines the size of the step. Choosing an appropriate α value takes some forethought. If it's too little, the program will take numerous incremental steps to arrive at the minimum. As a result, gradient descent may be sluggish.

If α is set very high, the algorithm may fail to converge and miss the minimum point. Worse, it may diverge, meaning it will move away from the minimum and

use an endless amount of time. The slope of the tangent line gets lower as θ it approaches the minimum until it hits zero. As a result, there's no need to change α , which may stay the same throughout the procedure.

6.3.7.3. Type of Gradient Descent

There are three types of gradient descent, each with a different amount of data used to compute the objective function's gradient. We create a trade-off between the precision of the parameter update and the time it takes to execute an update, depending on the amount of data.

1. Batch Gradient Descent

The gradient of the cost function with respect to the parameters θ is computed for the whole training dataset using vanilla gradient descent, also known as batch gradient descent.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta)$$

Batch gradient descent is complex and intractable for datasets that do not fit in memory, as we need to compute the gradients for the whole dataset to conduct just one update. Batch gradient descent also prevents us from updating our model in real-time. For convex error surfaces, batch gradient descent is guaranteed to converge to the global minimum, and for non-convex surfaces, to a local minimum.

2. Stochastic Gradient Descent

For each training example x_i and label y_i , stochastic gradient descent (SGD) conducts a parameter update. For big datasets, batch gradient descent performs unnecessary calculations by recalculating gradients for comparable samples before each parameter change. By making one update at a time, SGD eliminates redundancy. As a result, it is often faster, and it may also be used to study online.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta; x_i; y_i)$$

The volatility of the SGD allows it to reach new and maybe better local minima. This complicates the convergence to the precise minimum since SGD will continue to overshoot. SGD, on the other hand, has been demonstrated to have the same convergence behavior as batch gradient descent when the learning rate is gradually

reduced, virtually definitely converging to a local or global minimum for non-convex and convex optimization respectively.

3. Mini-batch Gradient Descent

Finally, mini-batch gradient descent combines the best features of both batch and stochastic gradient descent, performing an update for each mini-batch of n training samples.

$$\theta = \theta - \alpha \nabla_{\theta} J(\theta; x_{(i:i+n)}; y_{(i:i+n)})$$

It can employ highly optimized matrix optimizations common to state-of-the-art deep learning libraries to compute the gradient with respect to a mini-batch more efficiently. It can minimize the variance of parameter updates, which can lead to more steady convergence. Mini-batch sizes typically range from 50 to 256. However, this might vary depending on the application. When training a neural network, mini-batch gradient descent is often the algorithm of choice, and the name SGD is commonly used when mini-batches are utilized.

6.4. CONVOLUTIONAL NEURAL NETWORK

6.4.1. Overview

Convolutional neural networks (CNNs), a cultivated variety of neural networks explicitly built for this purpose, are introduced in this chapter. In the realm of computer vision, CNN-based architectures have grown so dominant that few people today would build a commercial product or compete in a competition involving picture recognition, object identification, or classification.

Around the 1980s, CNNs were developed and utilized for the first time. At the time, the best a CNN could do was detect handwritten digits. It was primarily used in the postal industry to read zip codes, PINs, and other similar information. The essential thing to understand about any deep learning model is that it takes enormous data and computational resources to train. Alex Krizhevsky decided in 2012 that it was time to revive the multi-layered neural network branch of deep learning. Researchers were able to revive CNNs due to the availability of enormous quantities of data, including ImageNet datasets with millions of tagged pictures and an abundance of computer resources [22].

The design of modern CNNs, as they are known informally, was influenced by biology, group theory, and much experimentation. When it comes to the task with a one-dimensional sequence structure, such as audio, text, and time series analysis, computer scientists frequently use CNNs. Since they are made up of neurons with learnable weights and biases, convolutional neural networks are pretty similar to conventional neural networks. Each neuron gets some inputs, does a dot product, and then executes a non-linearity if desired. From the raw picture pixels on one end to class scores on the other, the entire network still represents a single differentiable scoring function. They still have a loss function on the last layer, and we can use the same approaches we used to build ordinary neural networks.

First, we will go through the fundamental operations that form the foundation of all convolutional networks. The convolutional layers themselves, padding and stride, the pooling layers used to aggregate information across neighboring spatial areas, and a thorough examination of the topology of contemporary architectures are all included.

6.4.2. Algorithmic Framework

As we discussed, this section will be entirely devoted to the convolutional neural network. To understand CNN, we have to understand each layer and few concepts in detail. Multiple layers of artificial neurons make up convolutional neural networks. Artificial neurons are mathematical functions that calculate the weighted sum of many inputs and output an activation value, similar to their biological counterparts. Each convolutional neural network layer creates multiple activation functions passed on to the next layer when an image is input.

Convolutional neural networks use the fact that the input pictures to limit the design more logically. Unlike a conventional neural network, a CNN's layers have neurons organized in three dimensions: width, height, and depth. Fig. (6.22) depicts an overview of a convolutional neural network.

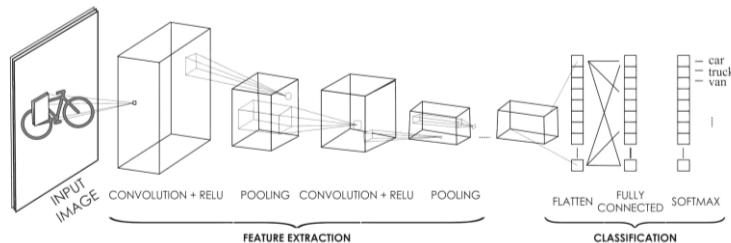


Fig. (6.22). A standard CNN architecture (Image adapted from towardsdatascience.com).

6.4.2.1. Layers

Different layers modify their inputs differently, and some layers are more suited for specific tasks than others.

A convolutional layer, for example, is commonly employed in models that operate with picture data. As the name implies, fully connected layers fully connect each input to each output inside their layer. Recurrent layers are employed in models that operate with time-series data.

6.4.2.1.1. Convolution layer

The convolutional layer is the pivotal component of a convolutional network. It is responsible for the majority of the computational effort [23]. To create an output, a convolutional layer cross-correlates the input and kernel and adds a scalar bias. The kernel and scalar bias are the two parameters of a convolutional layer. We usually randomize the kernels when we train models with convolutional layers, exactly like we would with a fully connected layer. Fig. (6.23) shows how an input image is being fed into a convolution layer.

A collection of learnable filters make up the convolution layer's parameters. Every filter is relatively small in size, yet it covers the whole depth of the input volume. While scanning the input in its dimensions, the convolution layer employs filters that conduct convolution operations. The filter size and stride are two of its hyperparameters. The final product is known as a feature map or an activation map. Any convolution neural network is built on the foundation of the convolution process. Every picture in CNN is represented as an array of pixel values.

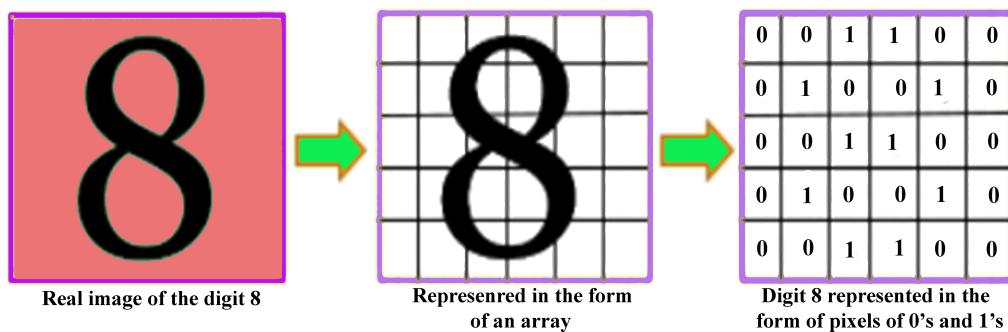


Fig. (6.23). Input image converts into the matrix before getting fed into CNN.

For example, a typical first-layer filter may be $10 \times 10 \times 3$, with 10 pixels of width and height and 3 pixels of depth since pictures have depth three (RGB color channels). Each filter is slide or 'convolved' over the width and height of the input volume and dot products between the filter's entries during the forward pass (Fig. 6.24). The input is computed at any location. As we move the filter over the input volume's width and height, we'll get a 2-D activation map with the filter's responses at every spatial point.

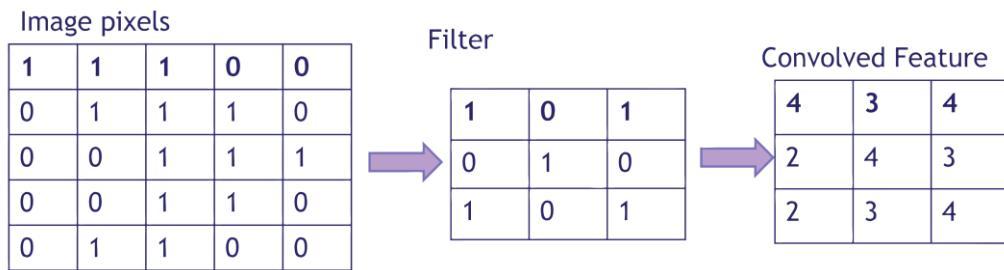


Fig. (6.24). Sample convolution operation on input image matrix using the filter to obtained convolved image.

1. Filter's Dimensions

A $F \times F \times C$ volume is an $F \times F$ filter that performs convolutions on an input of size $I \times I \times C$ and creates an output feature map of size $O \times O \times 1$ when applied to input with C channels.

2. Stride

The stride S specifies the number of pixels by which the window advances after each convolutional or pooling process.

3. Zero-padding

The act of adding P zeroes on each side of the input's bounds is known as zero-padding. This value can be manually entered or calculated automatically.

As it scans the input I in terms of its dimensions, the convolution layer employs filters that conduct convolution operations. The filter size F and stride S are two of its hyperparameters, and the result of the output is O . It is frequently helpful to calculate the parameters that a model's architecture will assess its complexity.

$$\begin{aligned} \text{Input size} &= I \times I \times C \\ \text{Output size} &= O \times O \times K \end{aligned}$$

$$\begin{aligned} \text{Number of parameters} &= (F \times F \times C + 1) \cdot K \\ \text{where } k &\text{ is the number of layers.} \end{aligned}$$

The output volume is controlled by three hyperparameters: depth, stride, and zero-padding. We may calculate the output volume's spatial dimension as a function of the input volume size (W), the convolution layer filter size (F), the stride (S), and the number of zero paddings utilized on the border (P). So the formula for determining how many neurons "fit":

$$(WF + 2P)/S + 1$$

Let's take an instance. Assume a 7x7 input with a 3x3 filter with stride 1 and pad 0 would yield a 5x5 output.

It is challenging to link neurons to all neurons in the preceding volume when dealing with high-dimensional inputs like pictures, as we saw previously. Instead, each neuron will be connected to only a tiny portion of the input volume. The receptive field of the neuron is a hyperparameter that describes the geographic extent of this connection. The depth of the input volume is always equal to the extent of connection along the depth axis. The connections are local throughout the width and height of the input volume but always full along with the whole depth.

The output of neurons linked to particular areas in the input will be computed by the convolution layer, which will calculate a dot product between their weights and a tiny region in the input volume that they are connected to.

6.4.2.1.2. RELU Layer

For finding features, an accurate picture is scanned in several convolutions and ReLU layers. Multiple convolutions, ReLU, and pooling layers are connected one after the other to extract features in each layer. The input feature map is created by scanning the input image several times.

The feature maps are fed into an activation function in the same way that they would be in a traditional artificial neural network. They are given into a rectifier function,

which returns zero if the input value is less than zero and returns the same input value if it is more than zero (Fig. 6.25).

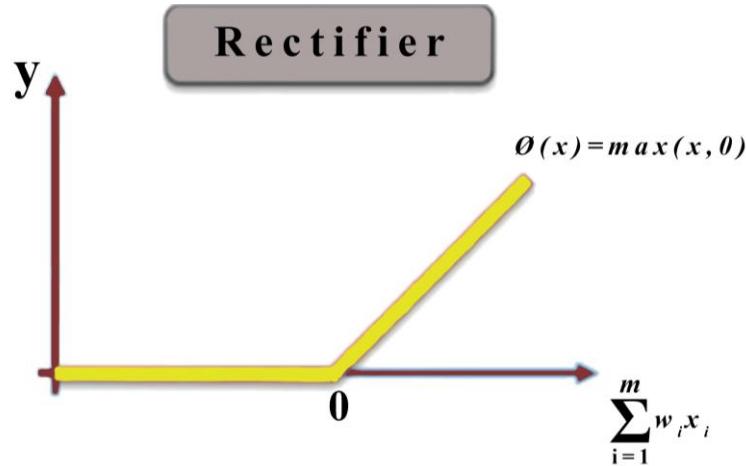


Fig. (6.25). RELU activation function being used after the convolution layer.

The rectifier function is frequently employed as the activation function in a convolutional neural network. It is used to improve the data set's nonlinearity. This may be described as a desire for a picture to be as gray-and-white as possible. The rectifier function successfully removes black pixels from the image and replaces them with gray pixels by eliminating negative values from the neurons' input signals.

6.4.2.1.3. Pooling Layer

A pooling layer is now applied to the corrected feature map. Pooling is a downsampling procedure that lowers the feature map's dimensionality. Like the convolutional layer, the pooling layer is responsible for shrinking the convolved feature's spatial size [24]. By decreasing the size, the processing power required to process the data is reduced. Average pooling and maximum pooling are the two forms of pooling. Fig. (6.26) shows the typical operation in a pooling layer.

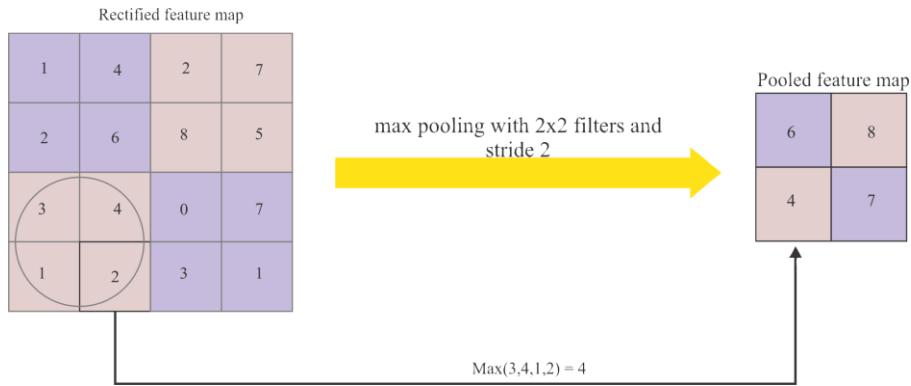


Fig. (6.26). Pooling layer with maxpool operation having 2×2 filter and stride size 2.

The pooling layer employs several filters to detect various aspects of the picture, such as edges, corners, body, feathers, eyes, and beaks. Using the max pooling operation, the pooling layer acts independently on each depth slice of the input and resizes it spatially. In the above picture, we see a pooling layer with filters of size 2×2 and a stride of 2. In this scenario, the max-pooling operation would take a maximum of two 2×2 values.

It accepts a volume of size: $W_1 \times H_1 \times D_1$

Requires two hyperparameters: the filter F and the stride S,

Producing a new volume having size: $W_2 \times H_2 \times D_2$

Where,

$$W_2 = \frac{W_1 - F}{S + 1}$$

$$H_2 = \frac{H_1 - F}{S + 1}$$

$$D_2 = D_1$$

The maximum pooling operation assigns the maximum value as the output. In contrast, the average pooling operation assigns the average value as the output, using the input components in the pooling window. One of the essential advantages

of a pooling layer is that it reduces the convolutional layer's excessive sensitivity to location. The padding and stride for the pooling layer can be specified. In combination with a stride greater than one, Max pooling can be utilized to shrink the spatial dimensions. The number of output channels in the pooling layer is the same as the number of input channels.

6.4.2.1.4. Flattening Layer

After you've acquired the pooled feature map, you'll need to flatten it. The whole pooled feature map matrix is flattened into a single column, which is then given to the neural network for processing. Flattening is the process of combining all of the pooled feature map's 2-D arrays into a single long continuous linear vector.

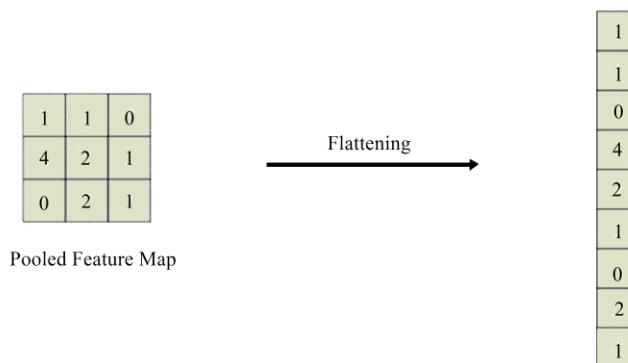


Fig. (6.27). Flatten layer: Converting the 2-D pooled matrix into 1-D vector.

The pooled feature map is transformed into a one-dimensional vector since it will be fed into an artificial neural network (Fig. 6.27). To put it another way, this vector will now serve as the input layer for a neural network.

6.4.2.1.5. Fully Connected Layer

The flattened feature map is then sent through a neural network after flattening. The input layer, the fully linked layer, and the output layer make up this stage. In ANNs, the completely connected layer is identical to the hidden layer, except it is fully linked in this case. The predictable classes are found in the output layer. The data is being sent *via* the network, and the prediction error is computed. To enhance the

forecast, the error is then backpropagated through the system. Fig. (6.28) depicts the fully connected layer in a CNN.

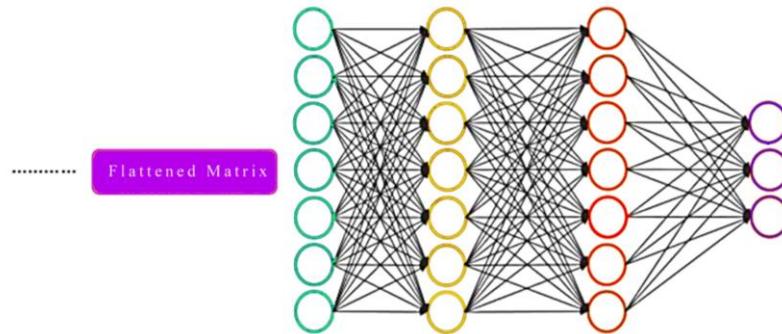


Fig. (6.28). Fully connected layer in CNN.

An artificial neural network is chained onto the current convolutional neural network at this full connection stage. In a convolutional neural network, the fully connected layer's job is recognizing specific characteristics in an image. The likelihood that the feature is present in the picture is represented by the neuron's numerical value thrown onto the next layer. As in standard neural networks, these neurons have complete connections to all activations in the preceding layer. As a result, their activations may be calculated using matrix multiplication and a bias offset.

6.4.3. Types of CNN

In the domain of convolutional neural networks, several architectures have been given names. Among them, some of the comprehensive architectures are given below:

1. LeNet

Yann LeCun pioneered the first successful uses of convolutional networks in the 1990s. The LeNet design, which was used to read zip codes, numbers, and other data, is the most well-known of these.

2. AlexNet

AlexNet, created by Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton, was the first effort to popularize Convolutional Networks in Computer Vision. The Network's design was quite similar to LeNet's, but it was deeper, more prominent, and included convolutional layers layered on top of each other.

3. VGGNet

Its primary contribution was to demonstrate that network depth is an essential factor in achieving high performance. Their best network has 16 convolution and fully connected layers and, more importantly, a highly homogenous design that performs just 3x3 convolutions and 2x2 pooling from start to finish. Their pre-trained model is offered in Caffe as a plug-and-play solution.

4. ResNet

Kaiming He *et al.* first introduced the residual network or ResNet. It has unique skip connections and makes extensive use of batch normalization. The design also lacks completely linked layers at the network's end. ResNets are the most advanced convolutional neural network models available today. They are the usual option for employing a convolutional neural network in practice.

5. GoogLeNet

The invention of an Inception module, which drastically decreased the network parameters, was its primary contribution. Furthermore, instead of using fully connected layers at the top of the convolutional neural network, this research utilizes average pooling, eliminating a significant number of parameters that do not appear to be necessary.

6.4.4. Programming Approach for Convolutional Neural Network

```
# 1. Import libraries

# Import tensorflow and keras
import tensorflow as tf
from tensorflow import keras
from keras.layers import Dense, Dropout, Flatten, Conv2D
from keras.models import Sequential, Model
```

```
from tensorflow.keras.utils import to_categorical

# Import other libraries
import numpy as np
import matplotlib.pyplot as plt

# 2. Import dataset

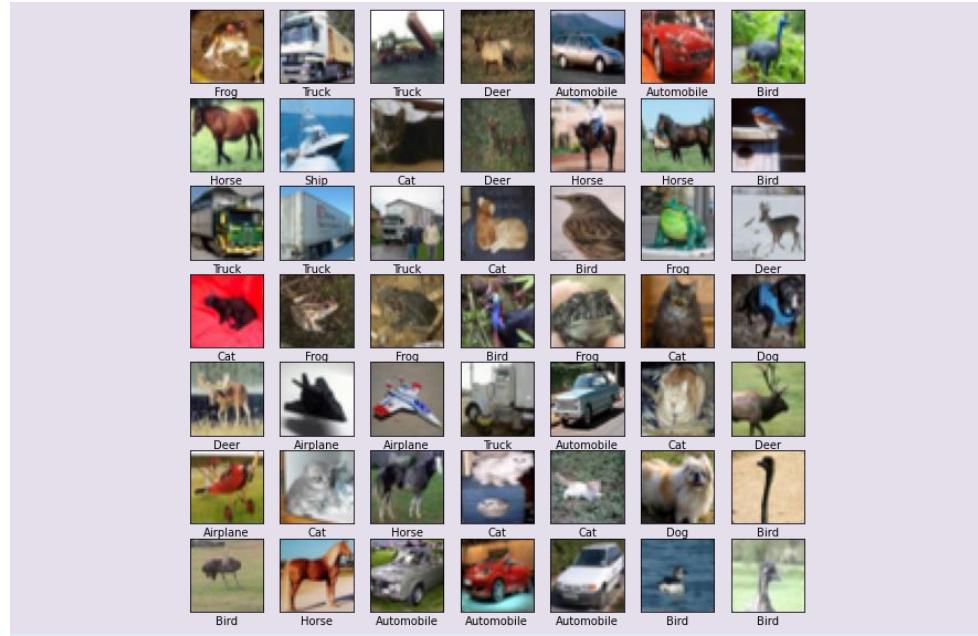
# Import cifar10 dataset
dataset = keras.datasets.cifar10

# Split dataset to train and test
(trainX, trainy), (testX, testy) = dataset.load_data()

# One hot encoding
trainY = to_categorical(trainy)
testY = to_categorical(testy)

Downloading data from
https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
170500096/170498071  [=====] - 2s
0us/step

# Set data labels
data_labels = ['Airplane', 'Automobile', 'Bird', 'Cat', 'Deer',
               'Dog', 'Frog', 'Horse', 'Ship', 'Truck']
# 3. Export example datas
plt.figure(figsize=(10,10)) # Predict size of print
for i in range(49):
    plt.subplot(7,7,i+1) # Print 7*7 datas
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    plt.imshow(trainX[i], cmap=plt.cm.binary) # Print images
    plt.xlabel(data_labels[int(trainy[i])]) # Print labels
plt.show() # Print conclusion
```



```
# 4. Make CNN model
CNN = Sequential() # Start making model
CNN.add(Conv2D(filters = 64, kernel_size = 3, activation = 'relu', input_shape = (32, 32, 3))) # 2D CNN layer with input layer
CNN.add(Conv2D(filters=64, kernel_size=3, activation='relu'))
CNN.add(Dropout(0.5)) # 50% Dropout
CNN.add(Flatten()) # Reduce dimension
CNN.add(Dense(100,activation='relu'))
CNN.add(Dense(10, activation='softmax')) # Fully connected layer with output layer
CNN.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # Compile model
CNN.summary() # Check model
```

Model: "sequential"

Layer (type)	Output Shape	Param
#		
conv2d (Conv2D)	(None, 30, 30, 64)	1792

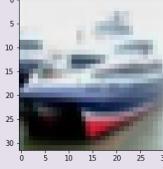
conv2d_1 (Conv2D)	(None, 28, 28, 64)	36928
dropout (Dropout)	(None, 28, 28, 64)	0
flatten (Flatten)	(None, 50176)	0
dense (Dense)	(None, 100)	5017700
dense_1 (Dense)	(None, 10)	1010
<hr/> <hr/>		
=====		
Total params: 5,057,430		
Trainable params: 5,057,430		
Non-trainable params: 0		
# 5. Learning model		
hist = CNN.fit(trainX, trainY, validation_data = (testX, testY), epochs=50, batch_size=32, verbose=1) # Learning		
Epoch 1/50		
1563/1563 [=====] - 54s 7ms/step -		
loss: 7.5301 - accuracy: 0.1213 - val_loss: 2.2952 -		
val_accuracy: 0.1151		
Epoch 2/50		
1563/1563 [=====] - 10s 6ms/step -		
loss: 2.2528 - accuracy: 0.1498 - val_loss: 2.0047 -		
val_accuracy: 0.2428		
Epoch 3/50		
1563/1563 [=====] - 10s 6ms/step -		
loss: 1.9544 - accuracy: 0.2672 - val_loss: 1.6887 -		
val_accuracy: 0.3762		
...		
...		
...		
Epoch 49/50		
1563/1563 [=====] - 10s 6ms/step -		
loss: 0.1764 - accuracy: 0.9599 - val_loss: 4.1499 -		
val_accuracy: 0.5796		
Epoch 50/50		

```

1563/1563 [=====] - 10s 6ms/step -
loss: 0.1954 - accuracy: 0.9578 - val_loss: 3.3307 -
val_accuracy: 0.5767

# 6. Test
result = CNN.predict_generator(testX[:25], steps=10) # Doing test
result = np.round(result) # Round result
plt.figure(figsize=(10,10))
for i in range(len(result)):
    print(i, ':') # Print data number
    plt.imshow(testX[i], cmap=plt.cm.binary) # Print images
    plt.show() # Print conclusion
    print("real : ", data_labels[np.argmax(testY[i])]) # Print real index
    print("predict : ", data_labels[np.argmax(result[i])]) # Print predicted index
    print('\n\n')

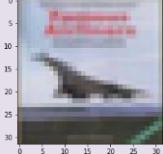
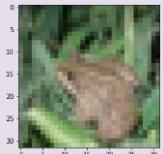
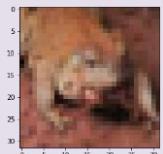
# Define test accuracy
_, score = CNN.evaluate(testX, testY, batch_size=32, verbose=0)
score = score * 100.0
print('# test accuracy: %.3f' % (score))

1 :

real : Ship
predict : Ship

2 :

real : Ship
predict : Ship

```

```
3 :  
  
real : Airplane  
predict : Ship  
  
4 :  
  
real : Frog  
predict : Frog  
  
5 :  
  
real : Frog  
predict : Frog  
  
# test accuracy: 57.670
```

6.5. RECURRENT NEURAL NETWORK

6.5.1. Overview

We have only come across two sorts of data so far: tabular data and imaging data. We unconsciously believed that our data came from the same distribution, which is not the case. If the words in this text were permuted randomly, it would be pretty impossible to understand their meaning. Similarly, optical frames in a movie, voice signals in a conversation, and internet surfing activity follow a strict sequence. Another difficulty emerges from the fact that we may be required to accept a sequence as an input and continue it. This is commonly used in time series analysis

to anticipate the stock market, a patient's fever curve, or the required acceleration for a race vehicle.

The fundamental property of a Recurrent Neural Network (RNN) is that it has at least one feedback link, allowing activations to loop back on themselves. This allows the networks to do temporal processing and learn sequences, such as sequence recognition or reproduction or temporal association or prediction. Recurrent neural network designs come in a variety of shapes and sizes. A conventional Multi-Layer Perceptron (MLP) with additional loops is one popular form. These can use the MLP's strong non-linear mapping capabilities while also having some memory.

In summary, while CNN's are good at processing spatial data, recurrent neural networks (RNNs) are superior at handling sequential data. RNNs use state variables to retain previous data and use it in conjunction with current inputs to determine current outputs.

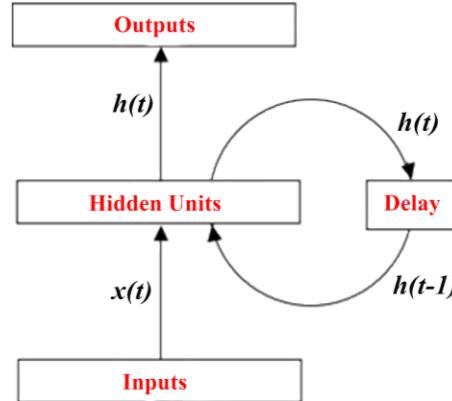
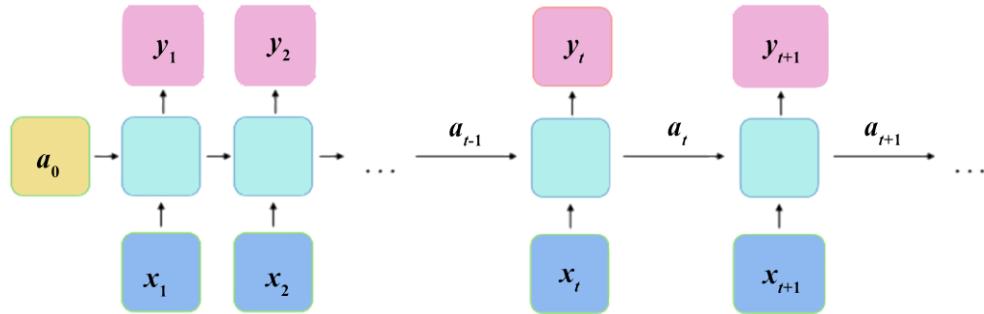
6.5.2. Algorithmic Framework

RNNs might learn to build a large number of tiny programs, each of which captures a nugget of information and runs in parallel, interacting to achieve very complex results. Despite some valiant attempts, such as Tony Robinson's voice recognizer, we were unable to fully harness the computing potential of RNNs for many years.

The MLP is the simplest type of RNN, with prior hidden units inserting back into the network.

To comprehend recurrent nets, you must first learn the fundamentals of feedforward nets. These networks get their names from how they transmit data *via* a sequence of mathematical operations carried out at the network's nodes. One sends data straight through, while the other loops it, and the latter is referred to be recurrent.

Recurrent networks use the current input sample they view as input and what they have seen in the past. Fig. (6.29) shows a typical working principle of an RNN unit. A recurrent net's choice at time step (t-1) impacts the decision it will make at time step t, one instant later (Fig. 6.30). So, recurrent networks have two sources of input: the present and the recent past, which combine to decide how they respond to input data.

**Fig. (6.29).** Simple RNN unit.**Fig. (6.30).** A classical architecture of the recurrent neural network.

In the classical architecture of the recurrent neural network, for each of the timestamps t , we can frame the equation of activation a_t as:

$$a_t = g_1(W_{aa}a_{t-1} + W_{ax}x_t + b_a)$$

And we can frame the output y_t as:

$$y_t = g_2(W_{ya}a_t + b_y)$$

where, W_{aa} , W_{ax} , W_{ya} , b_a , b_y are the coefficients with activation functions g_1 and g_2 .

Recurrent networks differ from feedforward networks in that they have a feedback loop related to previous judgments, absorbing their outputs as input moment after moment. Recurrent networks are frequently described as having memory. The sequence itself contains information, which recurrent networks employ to do jobs that feedforward networks can't. The hidden state of the recurrent network preserves that sequential information, which spans several time steps as it cascades forward to impact the processing of each subsequent sample.

We'll use mathematics to describe the process of transferring memory forward:

$$h_t = \emptyset(Wx_t + Uh_{t-1})$$

h_t is the hidden state at time step t . It is a function of the input at the same time step x_t , modified by a weight matrix W added to the hidden state of the previous time step h_{t-1} multiplied by its hidden-state-to-hidden-state matrix U , also known as a transition matrix similar to a Markov chain. The produced error will be sent to them *via* backpropagation and used to alter their weights until the error can no longer be reduced. The function \emptyset is either a logistic Sigmoid function or Tanh, depending on the sum of the weight input and hidden state, which is a common technique for condensing extremely big or very tiny values into a logistic space and making gradients practical backpropagation. Since this feedback loop happens at each time step in the series, each hidden state contains evidence of not just the preceding hidden state h_{t-1} , but also all those that came before it for as long as memory allows. A recurrent network will utilize the first character in a series of letters to assist in perceiving the second character.

6.5.2.1. Backpropagation Through Time

In the last section, we discussed forward and backward propagation. In an RNN, forward propagation is quite simple. In RNNs, backpropagation across time is a particular use of backpropagation. Backpropagation through time, or BPTT, is used in recurrent networks as an extension of backpropagation. Time is easily described by a well-defined, ordered series of calculations connecting one-time step to the next, all backpropagation needing to function. To get the dependencies among model variables and parameters, we must extend the computational network of an RNN one time step at a time.

Then, using the chain rule, we compute and store gradients *via* backpropagation. Since sequences can be rather long, the dependence can be quite long as well. For example, in a 5000-character sequence, the first token might substantially impact the token in the last place. This isn't computationally possible, and we'd have to do over 5000 matrix products to get to that elusive gradient.

Backpropagation through time performs gradient descent on an unfolded network. Let's look at some mathematics if we consider that any training sequence of a network starts at t_0 time and finishes at t_l time. Then the cost function can be framed as the summation over the standard error function for each time-step:

$$E_{total}(t_0, t_l) = \sum_{t_0}^{t_l} E_{se}(t)$$

Then the update of gradient descent weights from each time-step would be:

$$\begin{aligned}\Delta w_{ij} &= -\alpha \frac{\partial E_{total}(t_0, t_l)}{\partial w_{ij}} \\ &= -\alpha \sum_{t_0}^{t_l} \frac{\partial E_{se}(t)}{\partial w_{ij}}\end{aligned}$$

The partial derivative $\frac{\partial E_{se}}{\partial w_{ij}}$ have a contribution from various instances of each weight $w_{ij} \in \{w_{IH}, w_{HH}\}$ and rely on the input and hidden layer activations at preceding time-step (where w_{IH} and w_{HH} are the weights between the input to hidden units and hidden to hidden units, respectively). The error will now backpropagate through time across the network. In real-life applications, this technique shows enormous improvement.

Backpropagation through time is simply backpropagation applied to sequence models that have a hidden state. High matrices' powers might result in eigenvalues that are divergent or disappearing. Gradients that are bursting or disappearing are a result of this. Intermediate values are stored during backpropagation across time for faster calculation.

6.5.2.2. Need for More than Rnn: Vanishing and Exploding Gradient Problem

Recurrent networks, like most neural networks, are rather old. The vanishing gradient problem became a key roadblock for recurrent network performance. The gradient communicates the change in all weights concerning the change in error. If we don't know the gradient, we will not be able to modify the weights to reduce error, and the network will stop learning.

Deep neural networks' layers and time steps are related to one other by multiplication. Therefore derivatives are prone to disappearing or exploding. As they may be shortened or flattened, exploding gradients are very simple to fix. Vanishing gradients can become too tiny for computers or networks to understand - making it a more difficult challenge to tackle.

6.5.2.3. Types of RNN

Natural language processing and speech recognition are two domains where RNN models are commonly employed. Several forms of recurrent neural networks are accessible, depending on the applications, sources of input, and necessary output. Let's have a look at the different types:

1. One-to-one

One to one ($T_x=T_y=1$) RNN is also known as plain neural networks (Fig. 6.31). It is concerned with a fixed-size input to a fixed-size output, where the output is independent of the preceding output—for instance, application in image classification.

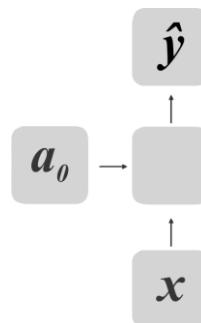


Fig. (6.31). One-to-one RNN.

2. One-to-many

One to Many ($T_x=1$, $T_y>1$) is a kind of RNN algorithm used when a single input yields numerous outputs (Fig. 6.32). Music creation is a simple illustration of how it may be used. RNN models are used in music generation models to produce a music piece from a single musical note.

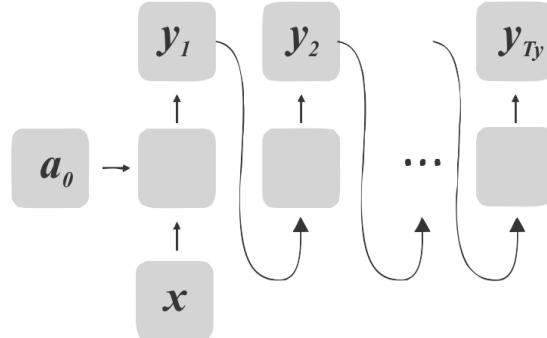


Fig. (6.32). One-to-many RNN.

3. Many-to-one

The many-to-one RNN architecture ($T_x>1$, $T_y=1$) is commonly encountered in sentiment analysis models (Fig. 6.33). This type of model is employed when numerous inputs are necessary to produce a single output, as the name implies—for instance, the sentiment analysis model for Twitter. Text input in that model provides its fixed sentiment.

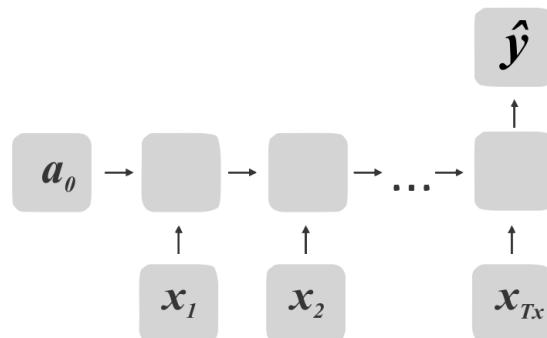


Fig. (6.33). Many-to-one RNN.

4. Many-to-many

Many-to-Many RNN ($T_x > 1$, $T_y > 1$) architecture accepts many inputs and produces multiple outputs, as is obvious, but there are two types of Many-to-Many models. Fig. (6.34) shows the basic many-to-many RNN architecture.

When the input and output layers are the same sizes, this type of many-to-many ($T_x = T_y$) is used. This may also be seen as every input having an output, with Named-entity Recognition being a famous example.

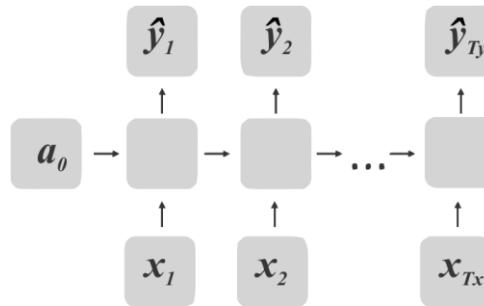


Fig. (6.34). Many-to-many RNN where T_x equals T_y .

5. Many-to-many (Bidirectional)

Many-to-Many architecture ($T_x \neq T_y$) may also be represented in models with different-sized input and output layers (Fig. 6.35), and machine translation is the most widespread use of this type of RNN architecture. Input and output sequences are synchronized. Since the recurrent transformation is constant and may be performed as often as we desire, there are no pre-specified limits on the lengths of sequences in any scenario. Consider video categorization, in which we want to classify each frame of the video.

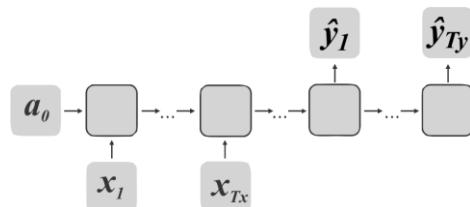


Fig. (6.35). Many-to-many RNN where T_x is not equal to T_y .

6.5.3. Hopfield Network

Neural networks were created to resemble the brain. The brain's memory, on the other hand, is based on associations. For example, even in a new setting, we can recognize a familiar face within 100-200 milliseconds. When we hear only a few music bars, we may recall a whole sensory experience, including noises and images. One item is frequently associated with another in the brain.

Pattern recognition issues are solved with multilayer neural networks trained using the back-propagation technique. However, we need a different sort of network to mimic the associative features of human memory: a recurrent neural network. From its outputs to its inputs, a recurrent neural network has feedback loops. The presence of such loops has a significant influence on the network's learning capabilities.

In the 1960s and 1970s, numerous academics were interested in the stability of recurrent networks. However, no one could predict which network would be stable, and several academics were skeptical that a solution could be found at all. Only in 1982, when John Hopfield established the physical idea of storing information in a dynamically stable network, was the challenge overcome. Fig. (6.36) shows single-layer n-neuron Hopfield network.

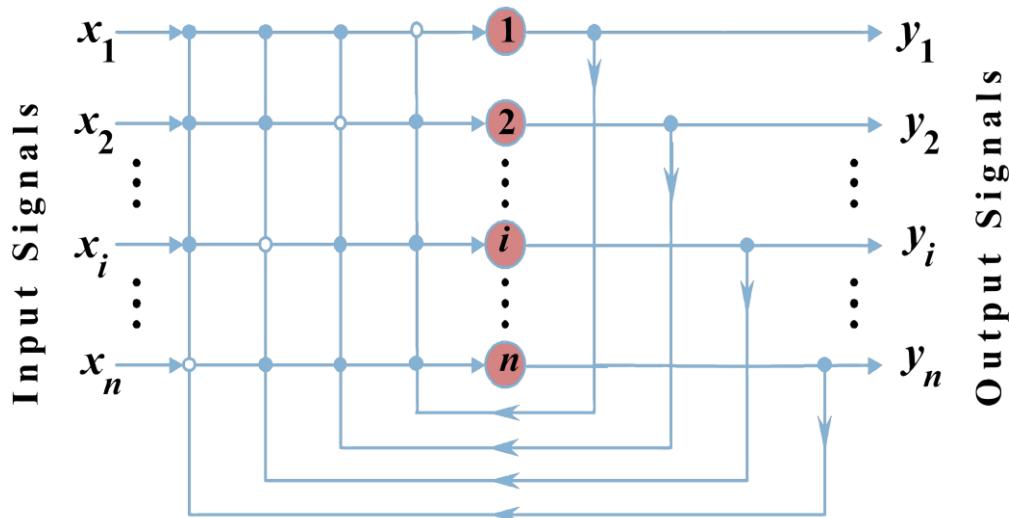


Fig. (6.36). Single-layer n-neuron Hopfield network.

Mathematically, we can express the weight matrix of the Hopfield network as:

$$W = \begin{bmatrix} 0 & w_{12} & w_{13} & \cdots & w_{1n} \\ \vdots & \ddots & & & \vdots \\ w_{1n} & w_{2n} & w_{3n} & \cdots & 0 \end{bmatrix}$$

Where input: $X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$ and $x_j = 1$ or -1

The Hopfield network employs McCulloch and Pitts neurons as its computational unit, with the sign activation function:

$$Y^{sign} = \begin{cases} +1, & \text{if } X > 0 \\ -1, & \text{if } X < 0 \\ Y, & \text{if } X = 0 \end{cases}$$

The current outputs of all neurons, $\mathbf{y}_1, \mathbf{y}_2, \dots, \mathbf{y}_n$, determine the Hopfield network's present state. The state vector may thus be defined as follows for a single-layer n-neuron network:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

Synaptic weights between neurons in the Hopfield network are often expressed as a matrix:

$$W = \sum_{n=1}^N Y_m Y_m^T - M \cdot I$$

Where, M is the number of states that the network must memorize, \mathbf{Y}_n is the n-dimensional binary vector, I is the $m \times m$ identity matrix, and superscript T indicates matrix transposition.

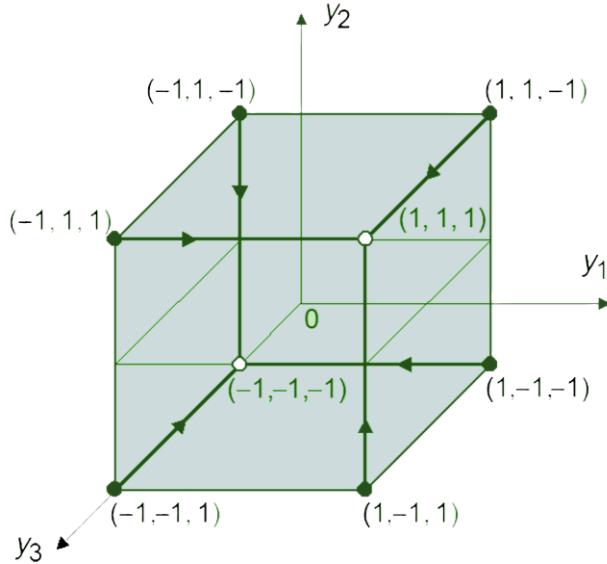


Fig. (6.37). Possible states for the three-neuron Hopfield network.

The weight matrix W , the current input vector X , and the threshold matrix q determine the stable state-vertex. After a few cycles, if the input vector is partially wrong or incomplete, the initial state will converge towards the stable state-vertex.

Assume that our network is expected to memorize two opposing states, $(1, 1, 1)$ and $(-1, -1, -1)$. Fig. (6.37) shows the possible states for the three-neuron Hopfield network. Thus,

$$Y_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \text{ and } Y_2 = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix}$$

$$\text{i.e., } Y_1^T = [1 \ 1 \ 1] \text{ and } Y_2^T = [-1 \ -1 \ -1]$$

where, Y_1 and Y_2 are 3-D arrays.

We know the weight calculation is done by $= \sum_{m=1}^N Y_m Y_m^T - M \cdot I$, so let us calculate the weight matrix with the given two 3-D arrays.

Let us consider the $m \times m$ identity matrix as 3×3 identity matrix I:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Now, we determine the weight matrix as:

$$\begin{aligned} W &= \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \cdot [1 \ 1 \ 1] + \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \cdot [-1 \ -1 \ -1] - 2 \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \end{aligned}$$

The network is then validated using input sequence X_1 and X_2 , those are equal to the output values Y_1 and Y_2 .

The Hopfield network is first activated by applying the input vector X. The actual output vector Y is then calculated, and the result is compared to the initial input vector X.

$$\begin{aligned} Y_1 &= \text{sign} \left\{ \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \\ Y_2 &= \text{sign} \left\{ \begin{bmatrix} 0 & 2 & 2 \\ 2 & 0 & 2 \\ 2 & 2 & 0 \end{bmatrix} \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \right\} = \begin{bmatrix} -1 \\ -1 \\ -1 \end{bmatrix} \end{aligned}$$

All six of the remaining states are volatile. Stable states might, on the other hand, attract states that are near to them. Instable states (-1, 1, 1), (1, -1, 1) and (1, 1, -1) are attracted to the basic memory (1, 1, 1). When compared to the fundamental memory, each of these unstable states defines a specific error (1, 1, 1). Instable states (-1, -1, 1), (-1, 1, -1) and (1, -1, -1) are drawn to the basic memory (-1, -1, -1). As a result, the Hopfield network may be used to rectify errors.

The maximum number of basic memories stored and retrieved accurately is referred to as storage capacity. M_{\max} is the maximum number of basic memories that may be stored in an n-neuron recurrent network. It can be denoted as $M_{\max} = 0.15n$.

6.5.4. Long Short Term Memory (LSTM)

The German academics Sepp Hoch Reiter and Juergen Schmid Huber suggested a variant of the recurrent network with so-called Long Short-Term Memory units, or LSTMs, to solve the vanishing gradient problem in the middle of 1990.

LSTMs aid in the preservation of error that can be transmitted backward in time and layers. They allow recurrent networks to learn across multiple time steps (over 1000) by keeping a more consistent error, creating a route to connect causes and effects remotely. Because algorithms are usually confronted by settings where reward signals are sparse and delayed, such as life itself, this is one of the critical difficulties facing machine learning and AI.

In a gated cell, LSTMs store data outside of the usual course of the RNN. Through open and close gates, the cell decides what to store and when to allow readings, writes, and erasures. Unlike digital storage on computers, these gates are analog, with element-wise multiplication by sigmoids in the range (0-1). Analog offers the benefit of being differentiable, and hence ideal for backpropagation, over digital. Fig. (6.38) shows the basic structure of a LSTM algorithm [25].

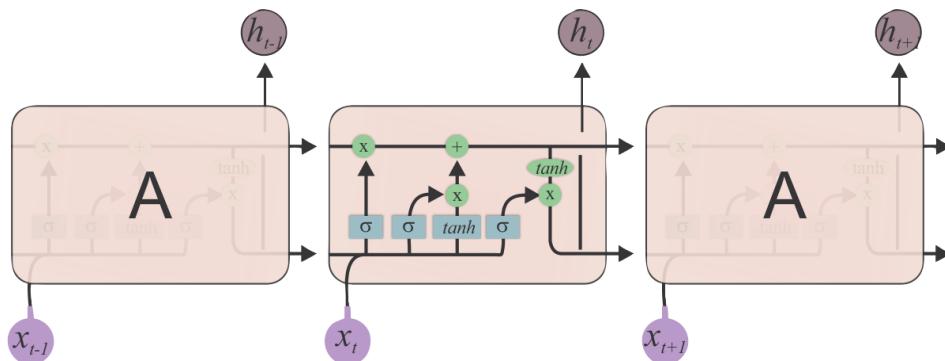


Fig. (6.38). A typical architecture of an LSTM model.

Those gates operate on the signals they receive, blocking or passing information according to its strength and import, which they filter with their own sets of weights, much like the nodes in a neural network. The learning process for recurrent networks adjusts such weights and the weights that modify input and hidden states. That is, by an iterative process of guessing, backpropagating error, and changing

weights via gradient descent, the cells learn when to allow data to enter, leave, or be eliminated.

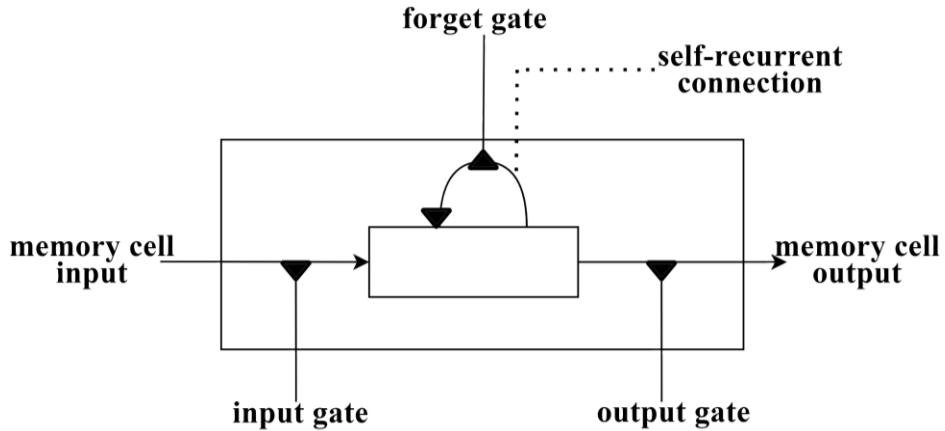


Fig. (6.39). The overview of the concept of gates in LSTM.

It's worth noting that addition and multiplication play different roles in the input transformation in LSTM memory cells. Instead of multiplying the current state by the incoming input to get the next cell state, they add the two, and that is the difference. For input, output, and forgetting, different sets of weights filter the data. Because if the forget gate is open, the current state of the memory cell is simply multiplied by one to propagate forward one more time step, the forget gate is represented as a linear identity function. Fig. (6.39) shows the different gates in a LSTM.

6.5.4.1. Few Concepts in LSTM

1. Cell State

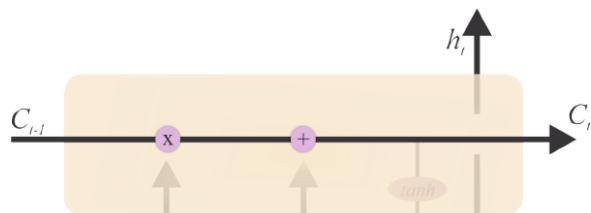


Fig. (6.40). Cell state in a LSTM.

The key idea of LSTMs is the cell state, which is a kind of conveyor belt used in factories (Fig. 6.40). It transfers the information from one side to another.

2. Forget Gate

LSTM may delete or add information to the cell state, which is carefully controlled via structures known as gates (Fig. 6.41). A sigmoid layer called the forget gate layer decides what information from the cell state should be discarded.

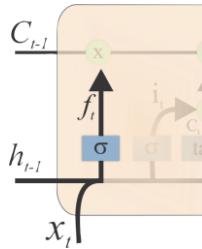


Fig. (6.41). Forget gate in a LSTM.

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

3. Input Gate

It determines what new data will be stored in the cell state. The input gate layer determines which values will be updated first (Fig. 6.42). The Tanh layer then generates a new vector of candidate values. Finally, merge the two to generate a state update.

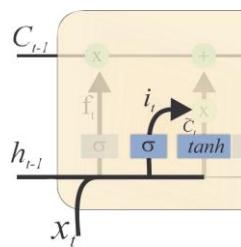


Fig. (6.42). Input gate in a LSTM.

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_c \cdot [h_{t-1}, x_t] + b_c)$$

4. Output Gate

The output gate determines the value of the next hidden state (Fig. 6.43). This state stores data from earlier inputs. The output is based on the cell state.

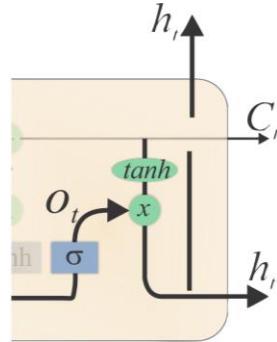


Fig. (6.43). Output gate in a LSTM.

$$\begin{aligned} O_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \\ h_t &= O_t * \text{Tanh}(C_t) \\ C_t &= f_t * C_{t-1} + i_t * \tilde{C}_t \end{aligned}$$

where $(f_t * C_{t-1})$ denotes forgetting last information, and $(i_t * \tilde{C}_t)$ denotes adding new information.

6.5.5. LSTM HYPER-PARAMETER TUNING

When manually tuning hyperparameters for RNNs, keep the following in mind:

1. Keep an eye out for overfitting, which occurs when a neural network "memorizes" the training data.
2. Regularization can help: examples of regularization methods are L1, L2, and Dropout.
3. Create a separate test set where the network will not be trained.

4. The more extensive the network, the more powerful it becomes; yet, it is also simpler to overfit.
5. Train throughout several epochs.
6. To know when to stop, evaluate the test set's performance at each epoch.
7. Use the Soft Sign activation function instead of Tanh for LSTMs.
8. Updates such as RMSProp, AdaGrad, and momentum are frequently used. AdaGrad also slows down the learning pace, which might be beneficial in some cases.

6.5.6. Programming Approach for Recurrent Neural Network

```
# 1. Import libraries
import numpy as np
import os
import matplotlib.pyplot as plt
from keras.models import Sequential
from keras.layers import Dropout
from keras.layers import SimpleRNN
from keras.layers import Flatten
from keras.layers import Dense
import scipy.io as sio
import scipy.io.wavfile
import tensorflow as tf
!pip install natsort
import natsort
from keras.utils.np_utils import to_categorical

# 2. Preparing Kaggle      https://medium.com/hyunjulie/%EC%
#BA%90%EA%B8%80%EA%B3%BC-%EA%B5%AC%EA%B8%80-colab-
%EC%97%B0%EA%B2%B0%ED%95%B4%EC%A3%BC%EA%B8%B0-6a274f6de81d

# Kaggle install
!pip install kaggle # Install kaggle
!pip install --upgrade --force-reinstall --no-
deps kaggle # Check newest version

# Upload json file
from google.colab import files
files.upload() # Kaggle.json import
```

```
# Json file import
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# Fixing permission
!chmod 600 ~/.kaggle/kaggle.json

# 3. Download dataset
!kaggle datasets download -d charlesaverill/imagenet-
voice # https://www.kaggle.com/charlesaverill/imagenet-
voice?select=inv_data
!unzip imagenet-voice.zip # Unzip dataset

# 4. Make dataset
path = os.listdir('inv_data') # Get file names
path = natsort.natsorted(path) # Sort file names

# Set data's num
num_class = 2
num_train = 30
num_val = 5
num_test = 5

# Make arrays
trainX = np.empty((1,5000,1))
valX = np.empty((1,5000,1))
testX = np.empty((1,5000,1))
trainY = list()
valY = list()
testY = list()

# Split data
for i in range(num_class):
    filepath = '/content/inv_data/' + path[i+5] + '/' # Set fi
le folders
    filenames = list()
    for j in range(num_train+num_val+num_test):
        filenames += [path[i+5] + '_' + str(j) + '.wav'] # Set
file names
    print('filenames',filenames)
    for t in range(num_train):
```

```

        sample_rate,dataframe = sio.wavfile.read(filepath + filenames[t], 'r') # Read wav file
        dataframe = dataframe[:5000] # Cut
        trainX = np.append(trainX, dataframe.reshape(1,-1,1), axis = 0) # Add to array
        trainY.append(i) # Make Y data
        for v in range(num_val):
            sample_rate,dataframe = sio.wavfile.read(filepath + filenames[v+num_train], 'r')
            dataframe = dataframe[:5000]
            valX = np.append(valX, dataframe.reshape(1,-1,1), axis = 0)
            valY.append(i)
        for te in range(num_test):
            sample_rate,dataframe = sio.wavfile.read(filepath + filenames[te+num_train+num_val], 'r')
            dataframe = dataframe[:5000]
            testX = np.append(testX, dataframe.reshape(1,-1,1), axis = 0)
            testY.append(i)

# Cut first row of X
trainX = np.delete(trainX, 0, axis = 0)
valX = np.delete(valX, 0, axis = 0)
testX = np.delete(testX, 0, axis = 0)

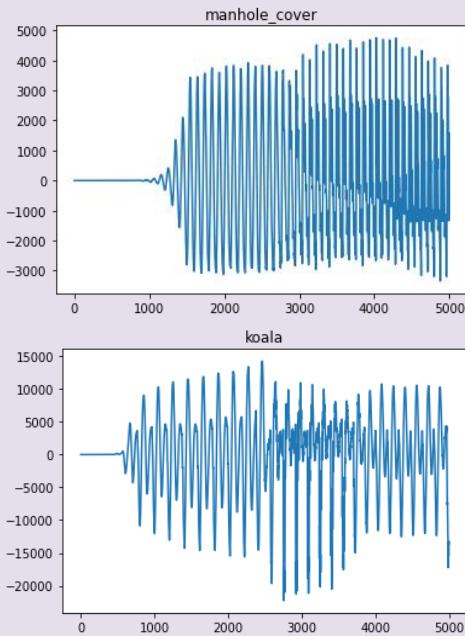
# One hot encoding
trainY = to_categorical(trainY)
valY = to_categorical(valY)
testY = to_categorical(testY)

# Make list Y to numpy
trainY = np.array(trainY)
valY = np.array(valY)
testY = np.array(testY)

# 5. Show example data
plt.figure(1)
plt.plot(trainX[1,:,:,:])
plt.title(path[5])

```

```
plt.figure(2)
plt.plot(trainX[11,:,:])
plt.title(path[6])
plt.show()
```



```
# 6. Make RNN model
RNN = Sequential()
RNN.add(SimpleRNN(256, input_dim=1, input_length=5000, return_sequences=True))
RNN.add(SimpleRNN(256, return_sequences = True))
RNN.add(Dropout(0.5))
RNN.add(Flatten())
RNN.add(Dense(2, activation='softmax'))
RNN.compile(loss='categorical_crossentropy', optimizer='adam',
', metrics=['accuracy'])
RNN.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
<hr/>		

simple_rnn_2 (SimpleRNN)	(None, 5000, 256)	66048
simple_rnn_3 (SimpleRNN)	(None, 5000, 256)	131328
dropout_1 (Dropout)	(None, 5000, 256)	0
flatten_1 (Flatten)	(None, 1280000)	0
dense_1 (Dense)	(None, 2)	2560002
<hr/>		
Total params:	2,757,378	
Trainable params:	2,757,378	
Non-trainable params:	0	

```
# 7. Learning model
hist=RNN.fit(trainX, trainY, validation_data = (valX, valY),
epochs=20, batch_size=32, verbose=1)

Epoch 1/20
2/2 [=====] - 22s 10s/step - loss: 3.3248 - accuracy: 0.4910 - val_loss: 28.7093 - val_accuracy: 0.7000
Epoch 2/20
2/2 [=====] - 20s 10s/step - loss: 1.6483 - accuracy: 0.9451 - val_loss: 28.2392 - val_accuracy: 0.7000
Epoch 3/20
2/2 [=====] - 21s 9s/step - loss: 0.2726 - accuracy: 0.9785 - val_loss: 39.3120 - val_accuracy: 0.4000
Epoch 4/20
2/2 [=====] - 21s 10s/step - loss: 2.8248 - accuracy: 0.9667 - val_loss: 53.3486 - val_accuracy: 0.6000
Epoch 5/20
2/2 [=====] - 20s 9s/step - loss: 5.5217e-08 - accuracy: 1.0000 - val_loss: 62.5805 - val_accuracy: 0.7000
...
...
Epoch 20/20
2/2 [=====] - 21s 10s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 137.2343 - val_accuracy: 0.6000
```

```
# 8. Test
_, score = RNN.evaluate(testX, testY, batch_size=32, verbose
=0)
score = score * 100.0
print('Test acc : %.3f' % (score))

Test acc : 60.000
```

CONCLUDING REMARKS

The reader is introduced to the most up-to-date, state-of-the-art deep learning algorithms in this chapter. Deep learning is a contemporary discipline of machine learning capable of recognizing the nature of data and comprehending the underlying patterns in it on its own. This chapter covered everything from neural network structure to advanced neural networks like convolutional neural networks and recurrent neural networks. It covered artificial, feed-forward, convolutional, and recurrent neural networks, with key topics like backpropagation, gradient descent, activation functions, and optimizations highlighted. This chapter will undoubtedly improve the readers' knowledge of advanced technologies by providing hands-on examples and a Pythonic approach to real-world applications.

Feature Engineering

Abstract: The chapter on feature selection techniques deals with most state-of-the-art feature selection techniques, which are being used alongside machine learning algorithms. The feature selection is a crucial element for the better performance of any machine learning algorithm. This chapter covers majorly two types of feature selection algorithms, namely, filter-based and evolutionary-based. This chapter covers two kinds of filter-based approaches in the filter-based algorithms, namely, hypothetical testing, such as t-test, z-test, ANOVA and MANOVA, and correlation-based such as Pearson's correlation, Chi-square test, and Spearman's rank correlation. This chapter also explains various methods such as genetic algorithms, particle swarm optimization, and ant colony optimization in evolutionary algorithms. For each of the algorithms, this chapter describes it in detail and the optimized algorithm for performing the feature selection approach.

Keywords: Feature Selection, Filter-based Approach, Wrapper-based Approach, Statistics, Hypothesis Testing, Correlation, Evolutionary Algorithm, T-Test, Z-Test, ANOVA, MANOVA, Pearson's Correlation Coefficient, Chi-Square Test, Spearman's Rank Correlation, Genetic Algorithm, Particle Swarm Optimization, Ant Colony Optimization.

7.1. INTRODUCTION

We encounter a vast number of features while working with big sets of data. The most difficult challenge is to choose the most particular and relevant feature from all the characteristics in a dataset. As a result, the feature selection techniques assist us in doing so.

The feature selection technique aids in the identification of important characteristics from a dataset. If an extensive data collection includes a input features, the feature selection method will choose certain important features, such as b . When selecting features, the condition is $b < a$.

In order to predict the target variable, feature selection approaches are used to minimize the number of input variables to those that are thought to be most beneficial to a model. The methods aid us in selecting the fewest number of features

in a data collection, which allows us to train a machine learning algorithm more quickly [26]. It helps in reducing a model's complexity and making it easier to understand. It develops a practical model with greater predictive potential by choosing the proper features to reduce over-fitting.

7.1.1. Types of Feature Selection

As this hit-and-trial technique is a time-consuming way for feature selection, other feature selection methods may be employed to speed up the process:

1. Filter Techniques

Filter techniques are commonly employed as a stage in the preprocessing process. Machine learning techniques have no impact on this feature selection. Instead, characteristics are chosen based on their connection with the outcome variable measured by various statistical tests.

2. Wrapper Techniques

We aim to utilize a subset of features in wrapper techniques and train a model with them. We determine whether to add or delete features from your subset based on the inferences drawn from the prior model. The problem may be simplified to a simple search problem. These approaches are generally highly time-consuming to compute.

3. Embedded Techniques

Filter and wrapper methods are combined in embedded methods. Algorithms with built-in feature selection techniques are used to build it. LASSO and RIDGE regression are two common examples of these techniques, both of which have built-in penalization mechanisms to reduce overfitting.

Before getting into individual algorithms of the filter-based approach, let us understand some proper feature selection techniques used in wrapper-based approaches. There are various feature selection approaches available, some of which treat the process as an art, others as a science, while, in reality, domain expertise and a disciplined approach are likely your best chance.

Wrapper techniques are disciplined approaches to feature selection that tie the feature selection process to the kind of model being produced, assessing feature subsets to identify model performance between features and then selecting the best performing subset [27].

The three primary techniques used to get the most optimal results under the wrapper method are:

1. Forward Selection
2. Backward Elimination
3. Bi-directional elimination

1. Forward Selection

Forward selection is an iterative approach in which no feature is included in the model at the start. We keep adding the feature that best improves our model in each iteration until adding a new variable no longer enhances the model's performance.

Forward feature selection begins by evaluating each individual feature and selecting the one that produces the highest performing algorithm model to consider the best set of features. The techniques depend on the assessment standards, such as accuracy, AUC, precision, recall, or error values. Following that, all potential combinations of the selected feature and the following feature are assessed. A second feature is chosen, and so on until the necessary predetermined number of features is chosen.

2. Backward Elimination

Backward elimination begins with all of the features and eliminates the least significant feature at each iteration, improving the model's performance. We repeat this process until no improvement is noticed when characteristics are removed. It is similar to forward selection in that it starts with the whole collection of features and works backward from there, eliminating features to find the best subset of a specific size.

3. Bi-directional Elimination

Until we need to delete previously added features, the forward selection and backward elimination procedures are helpful for feature selection. Both approaches are used in bi-directional elimination to pick the most relevant characteristics from a group of features. It functions similarly to a forward selection technique, but it also verifies previously added features' relevance. When the value of a previously introduced feature decreases, it is removed using the backward elimination approach. As a result, it is a hybrid of forward and backward elimination.

It is a greedy optimization method that seeks the best-performing feature subset. It generates models repeatedly, putting away the best or worst performing feature at each iteration. It builds the next model using the features on the left until all of the features are used up. The features are then ranked in order of their removal.

7.2. Filter-based Approach: Hypothesis Testing

Filter-based feature selection is the most frequent feature selection approach that one should be familiar with to build a decent model. The statistical relationship taught in school is used to choose the feature with the aid of this approach [26].

Applying statistical tests on inputs, given a predetermined output, is referred to as the filter-based technique. The objective is to figure out which traits are better at predicting the outcome. Multiple feature selection techniques are available in the filter-based feature selection module.

This section deals with the statistical hypothesis testing approaches under the filter-based methods. Hypothesis testing is a statistical procedure in which an analyst verifies a hypothesis about a population parameter. The analyst's technique is determined by the type of data and the purpose of the study. The use of sample data to assess the plausibility of a hypothesis is known as hypothesis testing. Such information might originate from a more significant population or a data-gathering method.

An analyst performs hypothesis testing on a statistical sample to prove the null hypothesis's plausibility. A hypothesis is tested by measuring and evaluating a random sample of the population being studied. All analysts use a random population sample to test two hypotheses: the null hypothesis and the alternative hypothesis.

The null hypothesis is generally a hypothesis of equivalence between population parameters; for example, the population mean might be equal to zero. A null hypothesis is basically the polar opposite of an alternative hypothesis. As a result, they are mutually exclusive, with only one of them being true.

The filter-Based feature selection module includes a number of statistical metrics for evaluating each attribute's worth. When we discuss the widely used statistical filter-bases methods, T-test, Z-test and ANOVA come in the first row. They are mainly used across the disciplines in recent researchers. This section gives a broad overview of each measure and how it is used [27].

7.2.1. T-test

A t-test is used to compare two big data sets, such as two large populations. A t-test is a statistical test that compares two sets of data and determines its difference. When we use a t-test, we get a t-score and a p -value. These numbers tell us how diverse each group is and how likely it is that this variation is due to chance or sampling error.

A t-test is an inferential statistic used to see if there is a significant difference in the means of two groups connected in some way. It is most commonly employed when data sets, such as those obtained by flipping a coin 100 times, are expected to follow a normal distribution and have unknown variances. A t-test is a hypothesis-testing technique that may be used to evaluate an assumption relevant to a population.

The t-test establishes the problem statement mathematically by taking a sample from each of the two sets and assuming a null hypothesis that the two means are identical. Specific values are computed and compared to the standard values using the appropriate formulae, and the presumed null hypothesis is accepted or rejected accordingly. If the null hypothesis is rejected, it means that the data readings are strong and are unlikely to be random. The t-test is only one of several tests that may be employed for this.

There are three types of t-tests, each of which is employed in distinct scenarios:

1. Independent Samples

The Independent Samples t-test compares the means of two independent groups to discover if there is statistical evidence that the population means are significantly different. This t-test is a parametric test.

▪ Formula

Let us call the two groups we're comparing x and y.

The means of groups X and Y are represented by m_x and m_y , respectively.

The sizes of groups X and Y are represented by n_x and n_y , respectively.

To determine if the means are different, the t-test statistic value is calculated as follows:

$$t = \frac{m_x - m_y}{\sqrt{\frac{S^2}{n_x} + \frac{S^2}{n_y}}}$$

Where, S^2 is a common variance estimator for the two samples. The given formula may be used to compute it. It is a common variance estimator for the two samples.

$$S^2 = \frac{\sum(x - m_x)^2 + \sum(x - m_y)^2}{n_x + n_y - 2}$$

Where $(n_x + n_y - 2)$ is the degree of freedom (d_f).

2. One Sample

The One-Sample t-test is a parametric test that examines if the sample mean differs from a known or hypothesized population mean statistically.

▪ Formula

Let X denote a set of n values with a mean of m and a standard deviation of S . The following formula is used to compare the observed mean μ of the population to a theoretical value:

$$t = \frac{m - \mu}{\frac{S}{\sqrt{n}}}$$

Where degree of freedom (d_f) = $n - 1$.

3. Paired Sample

The dependent sample t-test, also known as the paired sample t-test, is a statistical technique for determining if the mean difference between two data sets is zero. Each subject or object is measured twice in a paired sample t-test, resulting in pairs of observations. Case-control studies and repeated-measures designs are common uses for the paired sample t-test. Assume you want to assess the efficacy of a company's training program. You might use a paired sample t-test to compare the performance of a group of employees before and after they have completed the program.

▪ Formula

Let X denote a set of n values with a mean of m and a standard deviation of S . The formula is:

$$t = \frac{m}{\frac{S}{\sqrt{n}}}$$

Where the degree of freedom (d_f) = $n - 1$.

7.2.1.1. Hypothesis

In research, a hypothesis is used to explain a phenomenon or anticipate a relationship. A hypothesis must fulfill four assessment criteria to be considered

valid. First, a predicted connection between variables must be stated. Second, it must be testable and verifiable, which means that researchers must be able to determine if a theory is correct or incorrect. A hypothesis must be a precise, testable, and predictable assertion that is supported by theoretical guidance and/or past evidence.

For the Independent and One Sample tests, the null hypothesis is that both groups' means are identical. The null hypothesis for the Paired Sample is that the pairs of differences between both tests are equal.

The Independent Samples t Test's null hypothesis (H_0) and alternative hypothesis (H_1) may be stated in two distinct but identical ways:

$H_0: \mu_1 = \mu_2$ ("the two-population means are equal")

$H_0: \mu_1 \neq \mu_2$ ("the two-population means are not equal")

Or, in another way, we can define as:

$H_0: \mu_1 - \mu_2 = 0$ ("the difference between the two population means is equal to 0")

$H_0: \mu_1 - \mu_2 \neq 0$ ("the difference between the two population means is not 0")

where the population means for groups 1 and 2 are μ_1 and μ_2 , respectively.

Hypothesis Testing's core ideas are pretty similar to this case. Until we uncover convincing evidence to the contrary, we assume the Null Hypothesis to be correct. The Alternate Hypothesis is then accepted. We also calculate the Significance Level (α), which is the probability of something happening. As a result, if α is smaller, rejecting the Null Hypothesis will require more proof. Table 7.1. depicts clearly the occurrence of each kind of error due to outcome of null hypothesis and alternate hypothesis.

Table 7.1. Table showing the occurrence of Type-1 or Type-2 based on the decision on the hypothesis.

Real Truth Decision Made	Null Hypothesis (H_0)	Alternate Hypothesis (H_1)
Null Hypothesis (H_0)	No Error	Type-2 Error
Alternate Hypothesis (H_1)	Type-1 Error	No Error

For example, a person is on judgment by court. There are two kinds of judgment errors: Type 1 error, which occurs when the judgment is given against the person while he is innocent, and Type 2 error, which occurs when the judgment is given in favor of the person while he is guilty.

7.2.1.2. Tables of T-distribution

There are two versions of the T-Distribution Table: one-tail and two-tails. The one-tail is used to evaluate situations having a defined direction and a fixed value or range (positive or negative).

7.2.1.3. T-score

The t-score is the ratio of the mean difference between two datasets and the difference within the dataset generated by the T-test. While the numerator is simple to compute, the denominator, depending on the data values involved, can be a little more complicated. The ratio's denominator is a measurement of dispersion or variability. As a result, the higher the score, the greater the differences between the datasets, whereas the lower the score, the greater the similarities. T-Score is sometimes called a t-value.

7.2.1.4. P-value

A p -value is generated in addition to a t-score. This p -value indicates the likelihood that the discrepancy was caused by chance. In most cases, a p -value of less than 5% (at 95% confidence interval) is necessary to reject the null hypothesis.

7.2.1.5. Degree of Freedom

The amount of independent pieces of information used to calculate an estimate is referred to as degrees of freedom (df). It is not nearly the same as the sample's total number of items. To calculate the df for the estimate, subtract 1 from the total number of elements. Assume you were looking for the average weight loss on a low-carb diet. You might employ 10 people, giving you 9 degrees of freedom ($10 - 1 = 9$).

7.2.2. Z-test

When the variances are known and the size of the sample is large, a z-test is used to evaluate whether two population means are different. In order to execute an appropriate z-test, the test statistic is expected to have a normal distribution, and other factors such as standard deviation should be known.

The z-test is a hypothesis test where the z-statistic is distributed normally. The z-test is best utilized for samples with more than 30. This is because the samples with more than 30 are assumed to be approximately (regularly) distributed according to the central limit theorem [28].

The null and alternative hypotheses and the alpha and z-score should all be given when doing a z-test. The test statistic should next be computed, followed by the findings and conclusion.

So, we can frame the formula as:

$$Z = \frac{\bar{X} - \mu}{\sigma/\sqrt{n}}$$

where, \bar{X} is the mean of the sample,
 μ is the mean of the population,
 σ = standard deviation of the population
 n = population size (sample size)

7.2.2.1. Z-score Mean

A z-statistic, also known as a z-score, indicates how many standard deviations a score generated from a z-test, which is above or below the mean population. It is a numerical dimension that describes the connection between a value and the mean

of a collection of values. When a Z-score is zero, the data point's score is the same as the mean score. A number that is one standard deviation from the mean has a Z-score of 1.0. Z-scores can be positive or negative, with a positive number indicating that the score is higher than the mean and a negative value suggests low.

7.2.2.2. What Exactly is the Central Limit Theorem?

The central limit theorem (CLT) says that as the sample size grows bigger, the sample distribution approaches a normal, provided that all samples are similar in size and independent of the population distribution shape. For the CLT to correctly forecast the features of a population, sample sizes of 30 or more are deemed sufficient.

7.2.2.3. When to Perform a Z Test?

In statistics, several distinct types of tests are employed (*i.e.*, f-test, chi-square test, t-test). A Z test would be used if:

- The number of people in your sample is more than 30 [29].
- Data points should be unrelated to one another. In other words, one data point is unrelated to or unaffected by another.
- Your data should be spread evenly. This isn't usually an issue with high sample numbers (above 30).
- Your information should be chosen randomly from a population, with each item having an equal chance of being chosen.
- If at all feasible, sample sizes should be equal.

When we either know the population variance or don't know the population variance but have a large sample size ($n \geq 30$), z tests are a statistical technique of evaluating a hypothesis. Based on this, the z-test is of two kinds: one-sample z-test and two-sample z-test:

1. One-Sample Z-test

When we wish to compare a sample mean to the population mean, we use the One-Sample Z-test.

2. Two-Sample Z-test

When we wish to compare the mean of two samples, we use the Two-Sample Z-test.

7.2.2.4. Difference Between T-test and Z-test

1. Z: When standard deviation or variance is provided, the Z-test determines if the averages of the two datasets are different from each other [30].

T: The t-test is a type of parametric test used to determine how the averages of two sets of data differ from one another when the standard deviation or variance is unknown.

2. Z: The population variance, often known as the standard deviation, is known in the case of the z-test.

T: The population variance, often known as the standard deviation, is unknown in the case of the t-test.

3. Z: In the z-test, all data points are independent, and Z follows a Normal Distribution with a mean of zero and a variance of one [30].

T: In the t-test, All data points are independent, and sample values must be correctly documented and taken.

4. Z: The sample size is large in the z-test.

T: The sample size is small in the t-test.

5. Z: Z-test uses the Normal distribution as a foundation.

T: The Student-t distribution is used in the t-test.

7.2.3. ANOVA

The analysis of variance (ANOVA) is a statistical method that divides a data set's observed aggregate variability into two parts: systematic variables and random factors. Random factors have no statistical impact on the provided data set, but systematic variables do. In regression studies, analysts employ the ANOVA test to assess the impact of independent factors on the dependent variable.

The Fisher analysis of variance, commonly known as ANOVA, is an extension of the t-tests and z-tests. An ANOVA test is used to determine whether or not the findings of a survey or experiment are significant. To put it another way, they assist you in deciding whether you should reject the null hypothesis or accept the alternate hypothesis [31].

The ANOVA test allows you to compare more than two groups simultaneously to see whether there's a link between them. The F statistic (also known as the F-ratio) results from the ANOVA formula allow examining several sets of data to identify the variability between and within samples.

The quantity of independent variables in your ANOVA test determines whether it is one-way or two-way.

- **One-way Analysis:** There is just one independent variable in a one-way analysis (with two levels). Using the F-distribution, an One-way ANOVA is used to compare two means from two separate groups. The equality of the two means is the null hypothesis for the test. As a result of the significant finding, the two means are unequal.

- **Two-way Analysis:** A Two-way ANOVA is an extension of the One-way ANOVA. There are two independent variables in a two-way analysis (it can have multiple levels). When you have one measurement variable and two nominal variables, do a two-way ANOVA. In other words, a two-way ANOVA is acceptable if your experiment contains a quantitative outcome and two categorical explanatory factors.

The mathematical formulation for F-score in ANOVA is complicated yet straightforward due to conditions of variations. The following formula is given for the One-way ANOVA:

$$F = \frac{MST}{MSE}$$

Where mst stands for mean square of treatments, mse is the mean square of error.

$$MST = SST / p - 1$$

$$MSE = SSE / N - p$$

$$SSE = \sum (n - 1) \sigma^2$$

Where, F = Anova Coefficient

SST = Total Sum of squares

SSE = Sum of squares due to error

p = Number of population

n = Total number of samples in a population (Sample Size)

σ = Standard deviation of the samples

N = Total number of observations

7.2.4. MANOVA

MANOVA is nothing more than an ANOVA with multiple dependent variables. It's similar to many other tests and studies in that the goal is to see if altering the independent variable changes the response variable.

The MANOVA expands on this analysis by considering several continuous dependent variables and grouping them into a weighted linear combination or composite variable. The MANOVA will determine if the newly generated

combination varies from distinct groups or levels of the independent variable. The MANOVA simply evaluates whether the independent grouping variable concurrently explains a statistically significant amount of variation in the dependent variable in this way [32]. In particular, ANOVA examines the differences in means between two or more groups, whereas MANOVA examines the differences between two or more vectors of means.

It aids in the solution of a variety of research issues, including:

1. Do changes in the independent variables affect the dependent variables in a statistically significant way?
2. What are the relationships between the dependent variables?
3. What is the nature of the interactions between the independent variables?

7.2.4.1. Assumptions

- **Independent Random Sampling:** It assumes that the observations are independent of each other, that the sample is random, and that there is no pattern for selecting the sample.
- **Normality:** The data exhibits multivariate normality.
- **Measurement of the Variables:** It is based on the assumption that the independent factors are categorical and the dependent variables are continuous or scale variables.
- **Variance's Homogeneity:** There is no difference in variance across groups.
- **Multicollinearity's Absence:** There cannot be too much correlation between the dependent variables if there isn't multicollinearity.

7.2.4.2. Exceptional Situations

The inclusion of within-subjects independent variables and uneven sample sizes in cells are two exceptional situations in MANOVA.

- **Unequal sample sizes -** When cells in a factorial MANOVA have various sample sizes, the sum of squares for effect plus error does not equal the entire sum of squares, just like in ANOVA. As a result, the main effect and interaction tests are linked. In MANOVA, SPSS corrects uneven sample sizes.

- Within-subjects design - When a researcher assesses multiple dependent variables at various times, problems occur. This scenario can be regarded as a within-subject independent variable with as many levels as the number of times or individual dependent variables for each event.

7.2.4.3. MANOVA vs. ANOVA

1. Benefit of ANOVA Over MANOV

- a) MANOVA is a lot more complicated than ANOVA, making it challenging to figure out which independent factors impact dependent variables.
- b) With the addition of each additional variable, one degree of freedom is lost.
- c) As far as feasible, the dependent variables should be uncorrelated. If they are linked, the loss of degrees of freedom means that having more than one dependent variable on the test isn't worth it.

2. Benefit of MANOVA over ANOVA

- a) MANOVA allows you to test multiple dependent variables at the same time.
- b) Type I mistakes are protected by MANOVA.

7.3. FILTER-BASED APPROACH: CORRELATION

Filter-based feature selection approaches assess the correlation or dependency between input variables that may then be filtered to identify the most relevant features using statistical measures. The relationship between each input variable and the goal variable is evaluated using statistics, and the input variables having the strongest association with the target variable are selected. Although the choice of statistical measures depends on the data type of both the input and output variables, these approaches can be quick and successful.

The selection of filter features is frequently based on correlation type statistical measurements between input and output variables. As a result, the statistical measures used are heavily influenced by the data types that are changeable. Numerical and categorical variables are common data types. However, each can be

split further, such as integer and floating-point for numerical variables and boolean, ordinal, or nominal for categorical variables.

7.3.1. Correlation Analysis

The correlation test is used to determine the strength of a relationship between two variables.

Correlation analysis may be done in a variety of ways:

- **Parametric Correlation:** A parametric correlation is used to determine how linearly two variables are related. The data distribution affects this correlation technique.
- **Non-parametric Correlation:** Non-parametric correlations are ranked-based correlations such as Kendall (Tau) and Spearman (rho) coefficients.

Always keep in mind that the most favored correlation approach is a parametric correlation.

7.3.2. Pearson's Correlation

Pearson's correlation is a linear regression correlation coefficient. If you're new to statistics, you'll almost certainly start with Pearson's R. In reality, people generally mean Pearson's when people talk about correlation coefficients. Test statistics are used to determine the statistical link, or association, between two continuous variables. Since it is based on covariance, it is recognized as the best approach for assessing the connection between variables of interest. It provides information on the amount and direction of the relationship's connection or correlation.

The "product-moment correlation coefficient" or simply "correlation" is another name for the Pearson correlation. Pearson correlations are only helpful when dealing with numerical variables. The correlation coefficient formulae are used to determine the strength of a link between two sets of data.

$$r = \frac{\sum(x - m_x)(y - m_y)}{\sqrt{\sum(x - m_x)^2 \sum(y - m_y)^2}}$$

where, x and y are two vectors (data) of size n , m_x and m_y are the means of x and y .

7.3.2.1. Correlation Coefficient

Correlations might imply causal relationships or not. In the other direction, causal relationships between two variables may or may not result in a correlation between the two variables. Outliers can significantly influence correlations; a single unexpected observation can have a significant impact on a correlation. A cursory study of a scatterplot can quickly reveal such outliers.

The formula given above produce a value ranging from -1 to 1, where:

- A strong positive connection is indicated by a score of 1.
- A significant negative connection is indicated by a score of -1.
- A zero result implies that there is no link at all.

7.3.2.2. Assumptions

- Cases should be unrelated to one another.
- Two variables should have a linear connection with one another. A scatterplot may be used to determine this: plot the values of variables on a scatter diagram and see if the plot produces a straight line.
- The residuals scatterplot should resemble a rectangle.

7.3.2.3. The Cramer's V Correlation

The Pearson Correlation coefficient is equivalent to Cramer's V Correlation. Cramer's V is used to compute correlation in tables with more than 2×2 columns and rows, whereas Pearson correlation is used to assess the strength of linear connections. Cramer's V correlation ranges from 0 to 1. A value close to 0 suggests that the variables have a very low correlation. A Cramer's V value close to one implies a powerful link.

7.3.3. Chi-square Test

A hypothesis testing approach is the Chi-square test. χ is the Greek symbol known as Chi. Checking if observed frequencies in one or more categories match predicted frequencies is one of two popular Chi-square tests. For evaluating connections between categorical data, the Chi-Square statistic is widely employed. The null hypothesis is that the categorical variables in the population have no connection, they are independent [33].

When utilizing a cross-tabulation, the Chi-Square statistic is most often employed to evaluate Tests of Independence (also known as a bivariate table). The Test of Independence determines whether two factors are related by comparing the observed pattern of cell responses to the pattern that would be predicted if the variables were genuinely independent of one another. The researcher can determine if the observed feature set is significantly different from the predicted feature set by computing the Chi-Square statistic and comparing it to a critical value from the Chi-Square distribution.

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Where,

C is the degrees of freedom, (degrees of freedom are generally written as a subscript) O is the observed value, E is the expected value.

The summation sign indicates that you must calculate every data item in your data collection.

Chi-square tests are divided into two categories. For distinct reasons, both employ the chi-square statistic and distribution:

- **Chi-square Goodness of Fit:** The chi-square goodness of fit test assesses whether sample data represents the population.

For example, it determines whether each type of candy has the same amount of pieces in each bag.

- **Chi-square Test for Independence:** A chi-square test for independence examines two variables to discover if they are linked in a contingency table. In a broader sense, it examines whether categorical variable distributions differ from one another.

For example, it determines whether movie-goers' snack purchases are connected to the sort of film they want to watch.

A low chi-square score indicates that your two sets of data are highly correlated. In principle, chi-square would be 0 if observed and anticipated values were equal, but this is unlikely to happen in practice. It is not as straightforward as it seems to determine if a chi-square test result is large enough to suggest a statistically significant difference.

7.3.3.1. Chi-Square P-values

A p -value is determined using a chi-square test. The p -value indicates whether or not your test findings are significant [34]. You'll need two pieces of information to run a chi-square test and determine the p -value:

- Degrees of freedom: It's simply the number of categories divided by one.
- The alpha value level (α): The most common α level is 0.05 (5%), however other values such as 0.01 or 0.10 are also possible. The researcher himself chooses this.

7.3.3.2. Algorithm for Chi-square Test

You conduct the same analysis procedures for the Chi-square goodness of fit test and the Chi-square test of independence mentioned below.

1. Before you start collecting data, define your null and alternative hypotheses.
2. Decide on the alpha value. This entails determining how much of a chance you're prepared to take on the possibility of coming to the erroneous decision. Let's say you want to test for independence with $\alpha=0.05$. You have opted to take a 5% chance of concluding that the two variables are independent when they are not.

3. Check for any mistakes in the data.
4. Examine the test's assumptions.
5. Carry out the test and come to a decision.

7.3.3.3. Use of Chi-square Test

In statistics, the chi-squared distribution can be used for a variety of purposes, including:

1. Estimate a population's standard deviation of a normal distribution from a sample standard deviation using confidence intervals.
2. Two categorization criteria for qualitative variables are independent.
3. Categorical variables and their relationships (contingency tables).
4. When the underlying distribution is normal, sample variance analysis is used.
5. Differences between predicted and observed frequencies are tested using deviations (one-way tables).

7.3.4. Spearman's Rank Correlation

Spearman's rank correlation is a rank-based correlation coefficient. It's a non-parametric test for determining the strength of a relationship between two variables. When the variables are assessed on a scale that is at least ordinal, the Spearman rank correlation test is the proper correlation analysis since it makes no assumptions about the data distribution. The Spearman's rank correlation coefficient is a method for determining the degree and direction (positive or negative) of a link between two variables.

Spearman's correlation measures the degree and direction of a monotonic relationship between two variables. Monotonicity has fewer constraints than a linear connection. A monotonic connection is one in which the value of one variable rises in lockstep with the value of the other variable or one in which the value of one variable declines in lockstep with the value of the other variable. The assumption of a monotonic connection in Spearman's correlation is not strictly true.

To put it another way, you may use Spearman's correlation to see if a non-monotonic relationship has a monotonic component.

The Spearman's rank correlation is determined using the formula:

$$\rho = 1 - \frac{6 \sum d_i^2}{n(n^2 - 1)}$$

ρ = Spearman rank correlation

d = difference in ranks of corresponding variable

n = total number of observations

7.3.4.1. Algorithm for Spearman's Rank Correlation

1. Make a table out of your data.
2. Compare and contrast the two data sets. Giving the rating '1' to the largest number in a column, '2' to the second largest value, and so on, achieves ranking. The column with the smallest value receives the lowest ranking. For both sets of measurements, this should be done.
3. The mean rank is assigned to scores that are tied.
4. Determine the rank difference (d): This is the difference in rank between the two values on each table row. The first value's rank is deducted from the second value's rank.
5. To eliminate negative values, square the differences and then add them together.
6. Using the equation stated above, calculate the coefficient. The result will always be between 1.0 and -1.0, indicating a perfect positive correlation and a perfect negative correlation, respectively.

7.4. EVOLUTIONARY ALGORITHMS

Evolutionary algorithms are a heuristic-based technique to address problems that are difficult to answer in polynomial time, such as NP-Hard problems and anything else that would take too long to analyze exhaustively. Given that you are aware of the concept of natural selection, the premise of an evolutionary algorithm is relatively straightforward. Initialization, selection, genetic operators, and termination are the four main phases of an evolutionary algorithm. Each of these phases generally corresponds to a different aspect of natural selection and thus makes it simple to modularize implementations of this algorithm category.

Since they preferably do not make any assumptions about the underlying fitness landscape, evolutionary algorithms frequently do well at approximating solutions to a wide range of issues. Explorations of microevolutionary processes and planning models based on cellular processes are the most common evolutionary algorithms applications to biological evolution modeling.

There are various kinds of evolutionary algorithms. However, to satisfy the need and requirements of the readers of this book, this section presents the three most essential evolutionary algorithms, which are widely used alongside machine learning algorithms, as a part of feature selection. Here, we will be learning genetic algorithm, particle swarm optimization, and ant colony optimization in the next section.

7.4.1. Genetic Algorithm

In the year 1962, the foundations of genetic algorithms were developed. John Koza utilized genetic algorithms to develop programs to fulfill specific tasks later in 1992. Genetic programming was the name he gave to his approach.

Genetic algorithms are adaptable algorithms that belong to big evolutionary algorithms. Natural selection and genetics are the driving forces behind this method. For issues like optimization and search, genetic algorithms are frequently used. These are innovative applications of random search aided by previous data to steer the search to a solution space area with higher performance.

Natural selection is simulated by genetic algorithms, implying that the fittest species that can adapt to changing environments would survive among all individuals. The principle of survival of the fittest aids in the solution of problems

for successive generations. Various populations of different species exist in each generation, and these species are further represented as a string of character/ or integer or float or bits. The chromosome in the genetic algorithm is comparable to this string.

7.4.1.1. Foundation of Genetic Algorithm

In a population, there are diverse genetic structures and chromosomal behavior due to different species. The genetic algorithm is based on them, and the following are examples of them:

1. Individuals are always competing for resources and mating.
2. If successful mating occurs, more offspring are produced.
3. The ability to pass down strong or superior offspring from generation to generation.
4. As a result, each succeeding generation is more matched to their surroundings.

7.4.1.2. Search Space

A search space is a location where all of the people in the surroundings are kept safe. Each individual is a solution to a specific challenge. A finite-length vector of components represents each individual. Genes are comparable to these changeable components. As a result, a chromosome is made up of many genes.

As previously stated, there is continual rivalry among individuals for resources and mating, and each individual is assigned a fitness score. This score reflects an individual's level of fitness. Individuals (also considered a chromosome in the genetic algorithm) with the highest fitness scores are sought. The genetic algorithm keeps track of a population of n individuals and their fitness scores. An individual with a strong fitness score has a better probability of mating. Selection is made among those with higher fitness scores based on their offspring's fitness scores. Since the space size is fixed, some solutions die and are replaced by new ones to collect new offspring. After the previous generation's mating is completed, a new generation begins. It is hoped that as generations pass, better solutions will emerge, while the least suited will perish.

Each new generation has, on average, better genes than prior generations' individuals. As a result, each successive generation has better partial solutions than the preceding one. The population has converged when the children produced have no substantial differences from those generated by preceding populations. The algorithm is claimed to have converged on a set of problem solutions.

7.4.1.3. Genetic Operators

This process is really divided into two parts: crossover and mutation. Following the selection of the top chromosomes, these members are utilized to produce the algorithm's next generation. New offspring are generated using the traits of the chosen parents, who combine the parents' qualities. Depending on the type of data, this can be challenging, but combining combinations and producing valid combinations from these inputs in most combinatorial situations is feasible. The possibility of offspring acquiring a mutation and the severity of the mutation is usually determined by a probability distribution.

The genetic algorithm is applied when a particular generation is generated. For the same, there are various operators. Fig. (7.1) illustrates the lifecycle of a typical genetic algorithm.

1) Initialization: We must first build an initial population of chromosomes before we can begin the algorithm. An arbitrary number of alternative solutions to the issue referred to as people will be present in the population. It is frequently generated at random. As the population represents a gene pool, it must have a varied set of individuals or chromosomes.

2) Selection Operator: This operator is used to the fittest individual for its genes to be passed down over multiple generations.

3) Crossover Operator: Two individuals were chosen after utilizing the selection operator. Both individuals are permitted to mate thanks to the crossover operator. Crossover locations are chosen at random. The genes at these crossover locations are then swapped, resulting in creating an entirely new individual (offspring). Crossover is done at a specific point, known as crossover point.

4) Mutation Operator: The basic concept is to introduce random genes into children to preserve population variety and avoid premature convergence. Mutation allows the flipping of one or more bits (depending on condition) to alter the

characteristics of the chromosome. The mutation is controlled using a hyperparameter called mutation rate. This mutation rate can be altered with different values to get a new set of offspring.

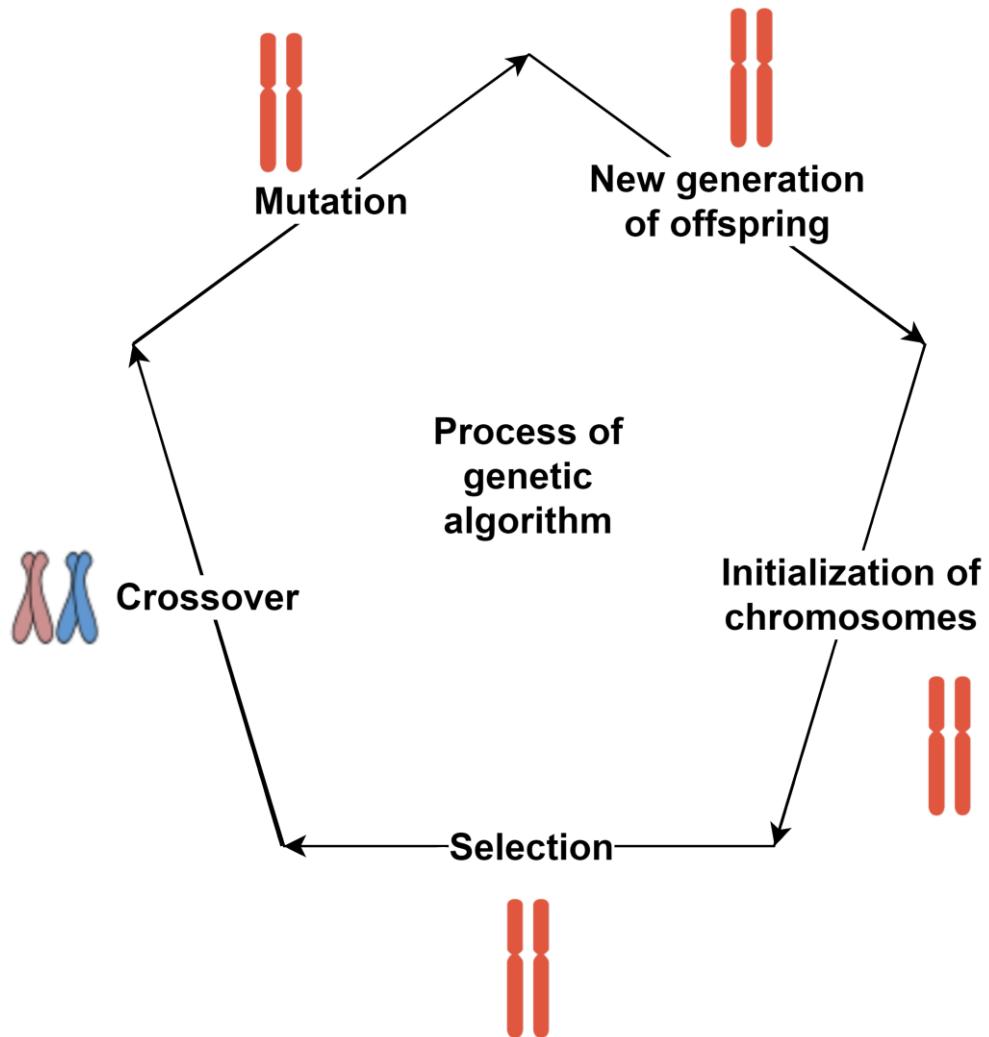


Fig. (7.1). A typical life cycle of chromosomes in a genetic algorithm.

7.4.1.4. Multi-objective Functions

Multiple objective or fitness functions can be used with a genetic algorithm. When utilizing several fitness functions, we end up with a group of optimum points rather than a single ideal point, making the procedure a little more complicated. The Pareto front is a collection of optimum solutions that comprises equally ideal chromosomes because no solution in the front dominates any other solution. Based on the context of the problem or some other parameter, a curve or line is then used to limit the set down to a single solution.

7.4.1.5. Termination

The algorithm must eventually come to an end. This generally happens in two ways: either the algorithm has achieved its maximum runtime, or the method has hit a performance threshold. A final solution is chosen and returned at this stage.

7.4.1.6. Algorithmic Framework

The following is a summary of the genetic algorithm:

- 1) Randomly initialize populations p .
- 2) Determine the population's fitness.
- 3) Repeat until convergence.
 - a) Choose parents from the general population.
 - b) Crossover and population generation.
 - c) Create a new population and perform mutations on it.
 - d) Determine the fitness of a new population.

7.4.2. Particle Swarm Optimization

Optimization is the process of determining the best values for a system's particular characteristics to meet all design criteria while keeping costs as low as feasible. Problems of optimization can be encountered in every branch of study.

Traditional optimization methods (deterministic algorithms) have several drawbacks, including:

- solutions based on a single principle
- convergence to local optimum
- problems with unclear search space

For these constraints, an improved optimization problem solution was required. Particle Swarm Optimization, Grey Wolf Optimization, Ant Colony Optimization, Genetic Algorithms, Cuckoo Search Algorithm, and so on are examples of optimization algorithms. Fig. (7.2) illustrates the lifecycle of a typical particle swarm optimization.

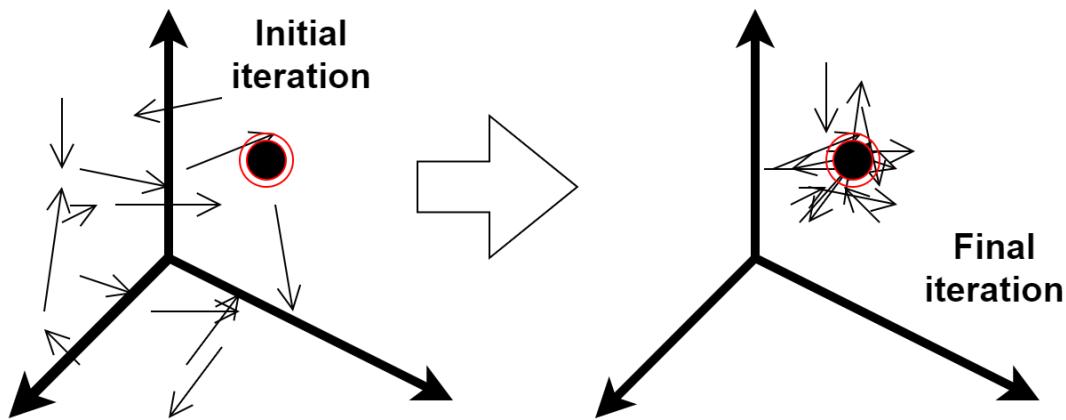


Fig. (7.2). A representative model showing the working behavior of PSO algorithm after n-iterations.

To demonstrate the Particle Swarm Optimization (PSO) algorithm's evolution and history, we first discuss the early basic model, namely the Boid (Bird-oid) model (Reynolds 1987). This model was created to imitate bird behavior and is also a primary source of the PSO algorithm. PSO has been used in a variety of research and application fields with great success.

PSO is a robust meta-heuristic optimization method that was inspired by natural swarm behavior like fish and bird schools. PSO is a simple social system

simulation. The PSO algorithm was created to graphically simulate the beautiful but unexpected dance of a flock of birds.

7.4.2.1. Particles

In nature, any of the bird's visible surroundings is confined to a specific range. Having several birds in a swarm, on the other hand, allows all of the birds in the swarm to be aware of the broader surface of a fitness function.

A set of particles (possible solutions) of the global minimum is known as a PSO in a search space. In this search space, there is only one global minimum. None of the particles knows where the global minimum is, but they all have fitness values that need to be optimized using the fitness function.

$$P_i^t = [x_{0,i}^t, x_{1,i}^t, x_{2,i}^t, \dots, x_{n,i}^t]$$

Let's take a minute to think about our particles before moving on to the PSO method. As you know, each of these particles represents a possible solution to the function that has to be reduced. Their coordinates in the search space define them.

The particles in the search space can then be defined at random. Each of these particles has a velocity that allows them to update their position over time to determine the global minimum.

$$V_i^t = [v_{0,i}^t, v_{1,i}^t, v_{2,i}^t, \dots, v_{n,i}^t]$$

In the search space, the particles have already been dispersed at random. After that, their velocity must be set. The velocity vector, which is defined by its speed in each direction, will be randomized once again. As a result, we refer to stochastic algorithms.

7.4.2.2. Swarms

PSO has a lot in common with evolutionary computing approaches like genetic algorithms. The system starts with a population of random solutions and then updates generations to look for optima. In contrast to genetic algorithms, however, PSO lacks evolution operators such as crossover and mutation. The distinction is in the manner in which the generations are updated.

$$P_i^{t+1} = P_i^t + V_i^{t+1}$$

Where $V_i^{t+1} = \omega V_i^t + c_1 r_1 (P_{best_{local(i)}}^t - P_i^t) + c_2 r_2 (P_{best_{global}}^t - P_i^t)$

Here, ωV_i^t is called inertia,

$c_1 r_1 (P_{best_{local(i)}}^t - P_i^t)$ is called cognitive,
 $c_2 r_2 (P_{best_{global}}^t - P_i^t)$ is called social (global)

Each particle's speed is stochastically accelerated towards its prior best position and the group's best solution across iterations in the search space.

Each particle's velocity is adjusted at the end of each iteration. The two best values determine this velocity found so far and are subject to inertia.

The first value represents the particle's greatest personal solution to date. The second is the best global answer the swarm of particles has discovered thus far. As a result, each particle has its best personal answer and the best global solution stored in its memory.

7.4.2.3. Optimization

The degrees of exploration and exploitation are influenced by inertia, cognitive, and social factors. Particles' ability to target the finest solutions discovered thus far is known as exploitation. Particles' capacity to assess the whole research space is known as exploration. It is targeted to find an outstanding balance between exploration and exploitation.

Random terms weigh the acceleration at each iteration. The weights r_1 and r_2 are used to modify cognitive and social acceleration stochastically. For each particle and iteration, these two weights, r_1 and r_2 are different.

The hyperparameter ω defines the swarm's capacity to shift direction. The inertia of the particles is related to the coefficient ω . The stronger the convergence, the smaller the coefficient ω . It's best to stay away from $\omega > 1$ since this might cause our particles to diverge.

The c_1 and c_2 coefficients are sometimes known as acceleration coefficients. The c_1 hyperparameter can be used to define the group's capacity to be affected by the best personal solutions discovered across iterations. The c_2 hyperparameter can be used to define the group's ability to be impacted by the best global solution identified across iterations. As a result, the coefficients c_1 and c_2 are complementary. Exploration and exploitation are both boosted when the two are combined.

We may take it a step further by updating coefficients as the iterations progress. Starting with a strong c_1 , strong ω , and weak c_2 to maximize search space exploration, we aim to converge to a weak c_1 , weak ω , and strong c_2 to utilize the best findings after exploration.

Compared to other approaches, PSO has been shown to produce superior outcomes faster and less expensive. It's also possible to parallelize it. Moreover, it does not take into consideration the gradient of the issue to be solved. To put it another way, unlike classic optimization approaches, PSO does not require a differentiable issue. The PSO algorithm comes in a variety of forms. Two factors often drive them. First, as PSO is near an evolutionary algorithm, hybrid versions with evolutionary capabilities are being developed. Second, by changing the hyperparameters, adaptive PSO can enhance performance.

7.4.3. ANT Colony Optimization

Marco Dorigo created Ant colony optimization (ACO) in the 1990s, inspired by ant colony foraging behavior. Ants are eusocial insects that want to survive and thrive as a colony rather than as individuals. They use voice, touch, and pheromones to communicate with one another. Pheromones are organic chemical substances produced by ants that cause individuals of the same species to react socially. These are substances that can function like hormones outside of the person's body who secret them, influencing the behavior of others who receive them. As most ants dwell on the ground, they leave pheromone trails on the soil surface that other ants may follow.

Ants reside in communal nests, and the basic idea of ACO is to watch the ants leave their homes to get food in the quickest possible time. Initially, ants roam around their nests at random in quest of food. This randomized search opens up multiple pathways from the nest to the food supply. Ants bring a portion of the food back

with the required pheromone concentration on its return journey, dependent on the quality and amount of the meal. The chance of the following ants choosing a specific path based on these pheromone trials would be a guiding element to the food source. This likelihood is depending on the pheromone concentration as well as the pace of evaporation. It is also worth noting that, because pheromone evaporation rate is a determining factor, the length of each route can be simply calculated [35].

To understand the scenario, let us assume an actual situation. Let us consider for simplicity that between the food source and the ant nest, just two potential pathways have been examined. Fig. (7.3) illustrates the working principle of ants and their locomotion with pheromones. The stages may be broken down into the following categories:

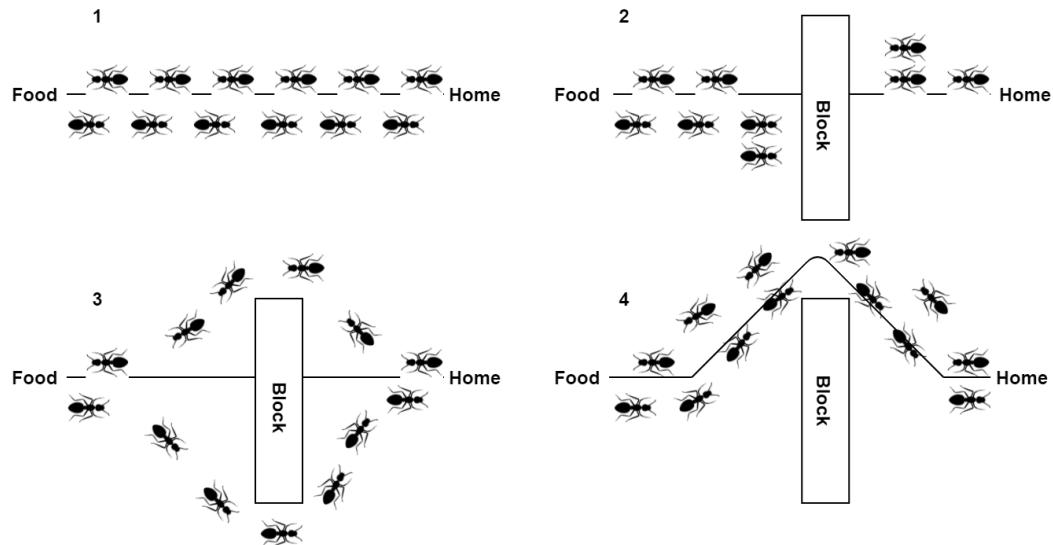


Fig. (7.3). Diagram showing the characteristics of ACO algorithm.

Stage 1: All ants have returned to their colony. In the environment, there is no pheromone content.

Stage 2: Ants begin their search along each trail with the same probability (0.5). The curved path is longer, and hence the time it takes for ants to reach the food source is longer.

Stage 3: The ants reach the food source faster by using the shorter path. They are now faced with a similar selection issue, but this time the likelihood of selection is higher due to a pheromone trail along the shorter path previously accessible.

Stage 4: As more ants return through the shorter path, the concentrations of pheromones rise as well. Furthermore, evaporation decreases the pheromone concentration in the longer path, lowering the likelihood of this path being chosen in subsequent phases. As a result, the colony as a whole begins to take the shorter way with more frequency. As a result, route optimization has been achieved.

7.4.3.1. Algorithmic Framework

The standard procedure to perform Ant Colony Optimization is given below:

1. At first, initialize all the necessary parameters and pheromone trials.
2. Until it is not terminated:
 - a) Generate the population of ants
 - b) Calculate the fitness values for each ant
 - c) Using the selection process, find the best solution
 - d) Update the pheromone trial using the formula:

$$\tau_{xy} = (1 - \rho)\tau_{xy} + \sum_k \Delta\tau_{xy}^k$$

ρ is the rate of evaporation of pheromone, τ_{xy}^k is the amount of deposited pheromone.

7.4.3.2. Application of Ant Colony Optimization

ACO's first applications were in the field of NP-hard combinatorial optimization problems. The majority of ACO research is still concentrated on this area, which is unsurprising. Routing in telecommunication networks is another use that was studied early in the history of ACO. The development of theoretical underpinnings and the application of the metaheuristic to new demanding situations are the focus of current ACO algorithm research.

CONCLUDING REMARKS

The chapter on feature selection approaches covered the most current feature selection strategies used in conjunction with machine learning algorithms. The selection of features is an essential factor in improving the performance of any machine learning system. The two types of feature selection algorithms discussed in this chapter were filter-based and evolutionary-based. This chapter discusses two types of filter-based algorithms: hypothetical testing, such as the t-test, z-test, ANOVA, MANOVA, and correlation-based, such as Pearson's correlation, Chi-square test, and Spearman's rank correlation. This chapter solely covered evolutionary methods, including genetic algorithms, particle swarm optimization, and ant colony optimization. This chapter goes through each method in-depth and the optimized algorithm for conducting the feature selection technique.

CHAPTER 8

Applications of Machine Learning and Deep Learning

Abstract: Until now, we have covered all aspects of machine learning and deep learning models and understood their internal architecture in detail. This chapter familiarizes the readers with the state-of-the-art real-life applications of machine learning and deep learning algorithms. The chapter will cover real-world applications from every corner of the recent advancements, starting from daily usage of face recognition to object detection. Not only that, but this chapter also explains the application of machine learning and deep learning in day-to-day life usage. Amid many applications, this chapter will cover the essential applications covering pattern recognition, video processing, medical imaging, and computational linguistics. The chapter presents the Python implementation of all the applications. This chapter also mentions some of the other critical real-world applications used in our daily life.

Keywords: Machine Learning, Deep Learning, Pattern Recognition, Video Processing, Medical Imaging Computational Linguistic, Classification, Tumor Detection, Neuroimaging, Face Recognition, Optical Character Recognition, Sentiment Analysis, Python, Tensor Flow, Keras.

8.1. INTRODUCTION

So far, we have looked at every aspect of machine learning and deep learning models and have a strong knowledge of their design. This chapter presents cutting-edge real-world applications of machine learning and deep learning techniques.

The chapter has looked at real-world applications from every corner of recent advancements, from everyday usage of face recognition to object detection. Not only that, but this chapter also discussed how machine learning and deep learning are applied in real-life situations.

Till now, we have discussed many algorithms of machine learning and deep learning. Although the explanation of every algorithm was supported by Python application on actual data, this chapter elaborates more on each algorithm. The availability of enormous datasets, along with improvements in algorithms and exponential increases in computer power, has resulted in an unprecedented surge of interest in the field of machine learning in recent years. Machine learning

techniques are now widely used for large-scale classification, regression, clustering, and dimensionality reduction problems involving high-dimensional input data. Other disciplines, such as image identification, have already been transformed by deep learning algorithms.

At first, we will cover machine learning applications and deep learning in pattern recognition, specifically for the most used daily applications such as face recognition, object detection, and optical character recognition [36]. After finishing all the image-related recognition applications, we will dig into video processing for identifying objects from a real-time or recorded video. An essential application of deep learning during the current time is a medical application, specifically medical imaging [37]. This chapter will teach how to apply machine learning and deep learning algorithms in neuroimaging to identify MRI and fMRI data patterns. Finally, we will visit another state-of-the-art application of deep learning, *i.e.*, the sentiment analysis using natural language processing, a computational linguistic part.

8.2. PATTERN RECOGNITION

In today's digital environment, the pattern is everywhere. A pattern can be physically detected or computationally observed *via* the use of algorithms. The automatic recognition of patterns and regularities in data is known as pattern recognition. Statistical data analysis, signal processing, picture analysis, information retrieval, bioinformatics, data compression, computer graphics, and machine learning are all areas where it may be used. It is the technique of applying a machine-learning algorithm to recognize patterns. The classification of data based on prior knowledge or statistical information collected from patterns and/or their representation is known as pattern recognition.

Pattern recognition has several characteristics. A pattern recognition system should be able to rapidly and accurately recognize patterns known to the user. It can adequately detect and categorize unknown things from various perspectives, allowing it to recognize and classify them. It can also detect patterns and things that are partially obscured. It automatically and rapidly identifies patterns.

8.2.1. Face Recognition

The technique of recognizing or validating a person's identification using their face is known as facial recognition. Patterns based on a person's facial features are

captured, analyzed, and compared. It is a way of recognizing or validating an individual's identification by looking at their face. Face recognition software can identify persons in pictures, videos, or in real-time. During police stops, officers may utilize mobile devices to identify persons.

Face recognition is one of the fields of machine learning that has been studied for a long time. Moreover, it has developed into a common and popular technology that can easily recognize faces even in our hands in recent years. In particular, it is used in mobile phones as a way to maintain security easily. Based on this reality, in this section, we will see how to distinguish faces with CNN and Avengers image sets, which show excellent ability to process image data.

8.2.1.1. Python Implementation

1. Technology used

- CNN
- Face recognition

2. Reference

- <https://medium.com/hyunjulie/%EC%BA%90%EA%B8%80%EA%B3%BC-%EA%B5%AC%EA%B8%80-colab-%EC%97%B0%EA%B2%B0%ED%95%B4%EC%A3%BC%EA%B8%B0-6a274f6de81d>
- <https://www.kaggle.com/rawatjitesh/avengers-face-recognition>
- <https://www.kaggle.com/ruslank1/brain-tumor-detection-v1-0-cnn-vgg-16/data>
- <https://www.tensorflow.org/tutorials/keras/classification>

3. Kaggle usage

- kaggle main page -> (click top-right circle) -> Account -> (down scroll) -> API -> Create New API Token

4. Description

- a) Import libraries
- b) Preparing Kaggle
- c) Download dataset
- d) Split data
- e) Dataset import
- f) print dates
- g) Make CNN model

```
h) Start learning
i) Save trained model
j) Import pre-trained model
k) Print result
l) Evaluate model
m) Print tested data

# 1. Import libraries
import matplotlib.pyplot as plt
import numpy as np
import os
import shutil
from pathlib import Path
from keras.preprocessing.image import ImageDataGenerator
from keras.layers import Input, Dense, Dropout, Flatten, Conv2D
from keras.models import Sequential, Model

# 2. Preparing Kaggle      https://medium.com/hyunjulie/%EC%BA%90%EA%B8%80%EA%B3%BC-%EA%B5%AC%EA%B8%80-colab-%EC%97%B0%EA%B2%B0%ED%95%B4%EC%A3%BC%EA%B8%B0-6a274f6de81d

# Kaggle install
!pip install kaggle # Install kaggle
!pip install --upgrade --force-reinstall --no-deps kaggle # Check newest version

# Upload json file
from google.colab import files
files.upload() # Kaggle.json import

# Json file import
!mkdir -p ~/.kaggle
!cp kaggle.json ~/.kaggle/

# Fixing permission
!chmod 600 ~/.kaggle/kaggle.json

# 3. Download dataset
!kaggle datasets download -d rawatjitesh/avengers-face-recognition # https://www.kaggle.com/rawatjitesh/avengers-face-recognition
!unzip avengers-face-recognition # Unzip dataset

path = './cropped_images/'
```

```

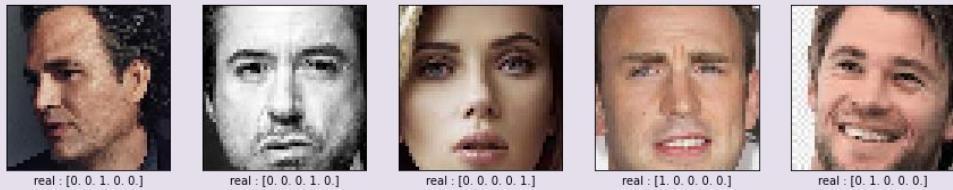
# 4. Split data      # https://www.kaggle.com/ruslankl/brain-
tumor-detection-v1-0-cnn-vgg-16/data
for data in os.listdir(path):  # For something in path
    if not data.startswith('.'): # If "data" is not empty
        num = len(os.listdir(path + data)) # num = file count
        s of folder(path + data)
        Path("./test/" + data.upper()).mkdir(parents=True, ex
ist_ok=True) # Make "test" directory
        Path("./train/" + data.upper()).mkdir(parents=True, ex
ist_ok=True) # Make "train" directory
        Path("./validation/" + data.upper()).mkdir(parents=True
, exist_ok=True) # Make "validation" directory
        for (n, name) in enumerate(os.listdir(path + data)):
# For something in folder(path + data)
            image = path + data + '/' + name # Make list of i
            mage data
            if n < 5: # Up to 50%
                shutil.copy(image, 'test/' + data.upper() + '/
' + name) # Copy files to "test" directory
            elif n < 0.8*num: # Up to 80%
                shutil.copy(image, 'train/' + data.upper() + '/
' + name) # Copy files to "train" directory
            else:
                shutil.copy(image, 'validation/' + data.upper()
+ '/' + name) # Copy files to "validation" directory

# 5. Dataset import
shape=48 # Define image's size
train = ImageDataGenerator(rescale=1./255).flow_from_directory(
    './train', target_size=(shape, shape), batch_size=5, color_m
ode = 'rgb', class_mode='categorical') # Import train data fr
om train folder
val = ImageDataGenerator(rescale=1./255).flow_from_directory(
    './validation', target_size=(shape, shape), color_mode = 'rgb'
, batch_size=5, shuffle = False, class_mode='categorical') # 
Import validation data from validation folder
test = ImageDataGenerator(rescale=1./255).flow_from_directory(
    './test', target_size=(shape, shape), color_mode = 'rgb', bat
ch_size=5, shuffle = False, class_mode='categorical') # Impor
t test data from test folder

Found 196 images belonging to 5 classes.
Found 53 images belonging to 5 classes.
Found 25 images belonging to 5 classes.

```

```
x, y = train.next() # Split train to image and label
# 6. Print data      https://www.tensorflow.org/tutorials/keras
/categorial_classification
plt.figure(figsize=(15,15)) # Predict size of print
for i in range(5):
    plt.subplot(1,5,i+1) # Print 1*5 data
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    plt.imshow(x[i], cmap=plt.cm.binary) # Print image
    plt.xlabel("real : " + str(y[i])) # Print label
plt.show() # Print conclusion
```



```
# 7. Make CNN model
model = Sequential() # Start making model
model.add(Conv2D(filters = 64, kernel_size = 3, activation = 'relu', input_shape = (shape, shape, 3))) # 2D CNN layer with input layer
model.add(Conv2D(filters=64, kernel_size=3, activation='relu')) # 2D CNN layer
model.add(Dropout(0.5)) # 50% Dropout
model.add(Flatten()) # Reduce dimension
model.add(Dense(100, activation='relu')) # Fully connected layer
model.add(Dense(5, activation='softmax')) # Fully connected layer with output layer
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # Compile model
model.summary() # Check model

Model: "sequential"
```

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 46, 46, 64)	1792

<code>conv2d_1 (Conv2D)</code>	(None, 44, 44, 64)	36928
<code>dropout (Dropout)</code>	(None, 44, 44, 64)	0
<code>flatten (Flatten)</code>	(None, 123904)	0
<code>dense (Dense)</code>	(None, 100)	12390500
<code>dense_1 (Dense)</code>	(None, 5)	505
<hr/>		
<hr/>		
Total params: 12,429,725		
Trainable params: 12,429,725		
Non-trainable params: 0		
<hr/>		
<hr/>		
batch_size = 32 # define model's batch_size		
# 8. Start learning		
hist = model.fit_generator(train, steps_per_epoch=train.n//batch_size, epochs=50, validation_data=val, validation_steps=5)		
# Learning		
# 9. Save trained model		
from keras.models import load_model		
model.save('new.h5') # Make pre-trained model and save		
<hr/>		
Epoch 1/50		
6/6 [=====] - 1s 347ms/step - loss: 3.0887 - accuracy: 0.1219 - val_loss: 1.6764 - val_accuracy: 0.0000e+00		
Epoch 2/50		
6/6 [=====] - 1s 244ms/step - loss: 1.6356 - accuracy: 0.3619 - val_loss: 1.6145 - val_accuracy: 0.1600		
Epoch 3/50		
6/6 [=====] - 1s 243ms/step - loss: 1.5955 - accuracy: 0.1890 - val_loss: 1.6091 - val_accuracy: 0.2800		
Epoch 4/50		

```
6/6 [=====] - 1s 219ms/step - loss:  
1.5344 - accuracy: 0.5540 - val_loss: 1.5736 - val_accuracy:  
0.3600  
...  
...  
...  
Epoch 49/50  
6/6 [=====] - 1s 241ms/step - loss:  
0.1175 - accuracy: 0.9029 - val_loss: 0.4393 - val_accuracy:  
0.8400  
Epoch 50/50  
6/6 [=====] - 1s 252ms/step - loss:  
0.0069 - accuracy: 1.0000 - val_loss: 0.5093 - val_accuracy:  
0.8400  
  
# 10. Import pre-trained model  
from keras.models import load_model  
model_import = load_model('new.h5') # Model import  
result = model_import.predict_generator(test, steps=10) # Doing test  
result = np.around(result) # Round result  
  
# 11. Print result  
print(test.class_indices) # Print class  
print(result) # Print result  
  
# 12. Evaluate model  
, score = model.evaluate(test, batch_size=32, verbose=0) # Calculate Accuracy  
score = score * 100.0  
print('Accuracy : %.3f' % (score)) # Print Accuracy  
  
{'CHRIS_EVANS': 0, 'CHRIS_HEMSWORTH': 1, 'MARK_RUFFALO': 2,  
'ROBERT_DOWNEY_JR': 3, 'SCARLETT_JOHANSSON': 4}  
[[1. 0. 0. 0. 0.]  
 [1. 0. 0. 0. 0.]  
 [1. 0. 0. 0. 0.]  
 [0. 1. 0. 0. 0.]  
 [0. 0. 1. 0. 0.]  
 ...  
 ...  
 [0. 0. 0. 0. 1.]  
 [0. 0. 0. 0. 1.]]  
Accuracy : 72.000
```

```

x, y = test.next() # Split test to image and label
# 13. Print tested data
plt.figure(figsize=(15,15)) # Predict size of print
for i in range(5):
    plt.subplot(1,5,i+1) # Print 1*5 data
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    plt.imshow(x[i], cmap=plt.cm.binary) # Print image
    plt.xlabel("real : " + str(y[i]) + "\npredict : " + str(resu
lt[i])) # Print label
    plt.show() # Print conclusion

```



8.2.2. Optical Character Recognition

The use of technology to recognize printed or handwritten text characters inside digital pictures of physical documents, such as scanned paper documents, is known as OCR (optical character recognition). OCR is a technology that examines a document's text and converts the characters into code that may be utilized for data processing. Text recognition is a term that is occasionally used to refer to OCR.

The electronic conversion of handwritten, printed, or image-only digital documents into a machine-readable and searchable digital data format is known as optical character recognition.

The way information is stored in a computer is a digital method consisting of 0's and 1's. However, since the environment in which humans live is based on analog, it was necessary to accompany a great deal of effort to input it into a computer. The need that has arisen as a result, is OCR. As machine learning is an old technique, there were cases where PDAs supported handwriting recognition from 30 years ago. Based on this, a recent recognition system applying advanced science and technology has overcome the existing low performance and efficiently and

accurately converted analog data into digital data. Therefore, this section aims to recognize it using the MNISTdataset, a collection of digits from 0 to 9.

8.2.2.1. Python Implementation

```
1. Technology used
• SVM
• MNIST dataset

2. Reference
• https://www.tensorflow.org/tutorials/keras/classification

3. Description
a) Import libraries
b) Download dataset
c) print dates
d) Make 2D image to 1D for training (Flatten)
e) Make
f) Training
g) Print result
h) Print tested data

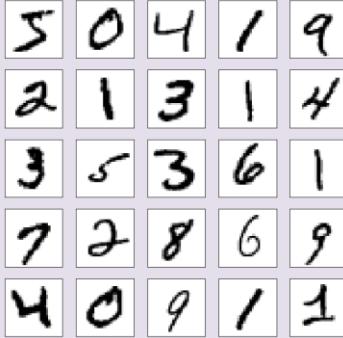
# 1. Import libraries
from tensorflow import keras
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn.metrics import accuracy_score

# 2. Download dataset
mnist = keras.datasets.mnist # Download mnist dataset
(trainX, trainY), (testX, testY) = mnist.load_data() # Split
data for train and test

# 3. Print data      https://www.tensorflow.org/tutorials/keras
/classification
plt.figure(figsize=(10,10)) # Predict size of print
for i in range(25):
    plt.subplot(5,5,i+1) # Print 5*5 data
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    plt.imshow(trainX[i], cmap=plt.cm.binary) # Print image
plt.show() # Print conclusion

(60000, 28, 28)
```

```
(60000, 28, 28)
...
(60000, 28, 28)


```

```
# 4. Make 2D image to 1D for training (Flatten)
train_X = trainX.reshape(60000, -1)
test_X = testX.reshape(10000, -1)

# 5. Make
model = svm.SVC()

# 6. Training
model.fit(train_X, trainY)

SVC(C=1.0, break_ties=False, cache_size=200, class_weight=None,
coef0=0.0, decision_function_shape='ovr',
degree=3, gamma='scale', kernel='rbf', max_iter=-1,
probability=False, random_state=None, shrinking=True,
tol=0.001, verbose=False)

# 7. Print result
result = model.predict(test_X)
accuracy_score(testY, result)

0.9792

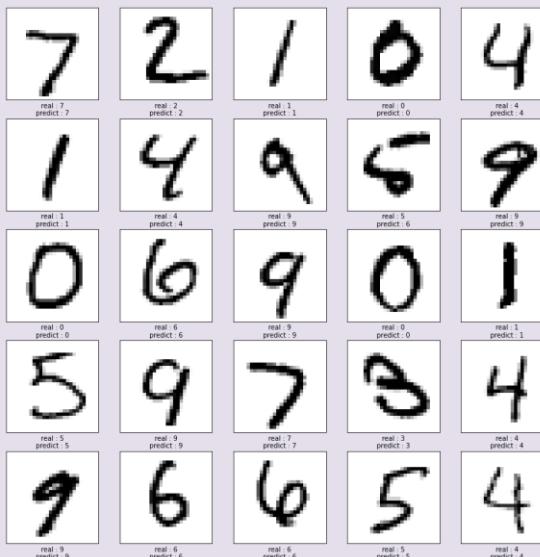
# 8. Print tested data      https://www.tensorflow.org/tutorials/
# keras/classification
plt.figure(figsize=(15,15)) # Predict size of print
for i in range(25):
    plt.subplot(5,5,i+1) # Print 5*5 data
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    print(testX.shape)
```

```
plt.imshow(testX[i], cmap=plt.cm.binary) # Print image
plt.xlabel("real : " + str(testY[i]) + "\npredict : " + str(result[i])) # Print label
plt.show() # Print conclusion
```

```
(10000, 28, 28)
(10000, 28, 28)
```

```
...
```

```
(10000, 28, 28)
(10000, 28, 28)
(10000, 28, 28)
```



8.2.3. Object Recognition

Object identification or recognition is a computer vision method that allows you to recognize things in pictures or movies. Deep learning and machine learning algorithms produce object recognition as a significant outcome. We can quickly notice individuals, objects, sceneries, and visual features when looking at a photograph or watching a video. The objective is to educate a computer to perform something naturally to humans: comprehend what an image conveys.

Object detection and object recognition are both methods for recognizing objects, but their implementation differs. The technique of detecting instances of things in pictures is known as object detection. Object detection is a subset of item

recognition in deep learning, where the object is identified and found in an image. Multiple items can be detected and found inside the same picture as a result of this. Machine learning allows you to pick the optimum mix of features and classifiers for learning when it comes to object identification. With only a few pieces of information, it may produce accurate findings.

It is a specific object that reflects in the camera. This is also one of the fields where machine learning is frequently used. Machine learning-based object recognition systems, such as YOLO, lead modern society as a primary driving force for unmanned systems. However, creating such a vision recognition system requires a lot of data, which has been recognized as a costly and laborious business. It is transfer learning that sheds light on such a situation. Transfer learning is an efficient machine learning technique that can achieve great results by training a small amount of data on an existing model. Therefore, this chapter aims to learn two cat and dog classes through transfer learning and output them as models.

8.2.3.1. Python Implementation

- 1. Technology used**
 - Transfer learning
 - Cat dog dataset usage
- 2. Reference**
 - <https://www.tensorflow.org/tutorials/keras/classification>
 - <https://cafe.daum.net/hyunsikahn/FPA7/12>
- 3. Dataset**
 - <https://www.kaggle.com/tongpython/cat-and-dog>
 - Download dataset, and split, upload the archive folder located on the main screen of Google Drive.
- 4. Description**
 - a) Import libraries
 - b) Import google drive
 - c) Dataset import
 - d) print dates
 - e) Import pre-trained dataset (VGG16)
 - f) Make additional learning course
 - g) Summation pre-trained and additional
 - h) Transfer learning
 - i) Save trained model
 - j) Import test data

```

k) Import pre-trained model
l) Print result
m) Evaluate model
n) Print tested data

# 1. Import libraries
import matplotlib.pyplot as plt
import numpy as np
from keras.preprocessing.image import ImageDataGenerator
from keras.applications.vgg16 import VGG16
from keras.layers import Dense, Dropout, Flatten
from keras.models import Sequential, Model

# 2. Import google drive
from google.colab import drive
drive.mount('/content/drive')

from google.colab import files

# 3. Dataset import
shape=48
train = ImageDataGenerator(rescale=1./255).flow_from_directory(
    '/content/drive/MyDrive/archive/training_set', target_size=(shape, shape),
    batch_size=5, color_mode = 'rgb', class_mode='categorical') # Import train data from train folder
test = ImageDataGenerator(rescale=1./255).flow_from_directory(
    '/content/drive/MyDrive/archive/test_set', target_size=(shape, shape),
    color_mode = 'rgb', batch_size=5, shuffle = False, class_mode='categorical') # Import test data from test folder

Found 402 images belonging to 2 classes.
Found 304 images belonging to 2 classes.

x, y = train.next() # Split train to image and label
# 4. Print data https://www.tensorflow.org/tutorials/keras/classification
plt.figure(figsize=(15,15)) # Predict size of print
for i in range(5):
    plt.subplot(1,5,i+1) # Print 1*5 data
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    plt.imshow(x[i], cmap=plt.cm.binary) # Print image

```

```
plt.xlabel("real : " + str(y[i])) # Print label
plt.show() # Print conclusion
```



```
# 5. Import pre-trained dataset (VGG16)
vgg = VGG16(include_top = False, weights = 'imagenet', input_shape = (shape,shape,3)) # Import VGG16 to vgg
for layer in vgg.layers:
    layer.trainable = False # All layer's weights to non-trainable = freezing

# 6. Make additional learning course
add = Flatten(name = 'flatten')(vgg.output) # Reduce dimension
add = Dense(4096, activation='relu')(add) # Fully connected layer
add = Dense(4096, activation='relu')(add) # Fully connected layer
add = Dropout(0.5)(add) # 50% Dropout
add = Dense(2, activation = 'softmax')(add) # Fully connected layer with output layer

# 7. Summation pre-trained and additional
model_new = Model(vgg.input, add) # Summarize vgg and add
model_new.summary() # Check model
model_new.compile(loss = 'categorical_crossentropy', optimizer = 'adam', metrics = ['accuracy']) # Compile model

Downloading data from
https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_ordering_tf_kernels_no_top.h5
58892288/58889256 [=====] - 1s
0us/step
58900480/58889256 [=====] - 1s
0us/step
Model: "model"
```

Layer (type)	Output Shape	Param #
<hr/>		
= input_1 (InputLayer)	[None, 48, 48, 3]	0
 block1_conv1 (Conv2D)	(None, 48, 48, 64)	1792
 block1_conv2 (Conv2D)	(None, 48, 48, 64)	36928
 block1_pool (MaxPooling2D)	(None, 24, 24, 64)	0
 block2_conv1 (Conv2D)	(None, 24, 24, 128)	73856
 block2_conv2 (Conv2D)	(None, 24, 24, 128)	147584
 block2_pool (MaxPooling2D)	(None, 12, 12, 128)	0
 block3_conv1 (Conv2D)	(None, 12, 12, 256)	295168
 block3_conv2 (Conv2D)	(None, 12, 12, 256)	590080
 block3_conv3 (Conv2D)	(None, 12, 12, 256)	590080
 block3_pool (MaxPooling2D)	(None, 6, 6, 256)	0
 block4_conv1 (Conv2D)	(None, 6, 6, 512)	1180160
 block4_conv2 (Conv2D)	(None, 6, 6, 512)	2359808
 block4_conv3 (Conv2D)	(None, 6, 6, 512)	2359808
 block4_pool (MaxPooling2D)	(None, 3, 3, 512)	0

<hr/>		
block5_conv1 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv2 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv3 (Conv2D)	(None, 3, 3, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten (Flatten)	(None, 512)	0
dense (Dense)	(None, 4096)	2101248
dense_1 (Dense)	(None, 4096)	16781312
dropout (Dropout)	(None, 4096)	0
dense_2 (Dense)	(None, 2)	8194
<hr/>		
=		
Total params:	33,605,442	
Trainable params:	18,890,754	
Non-trainable params:	14,714,688	
<hr/>		
batch_size = 32 # define model's batch_size		
# 8. Transfer learning		
hist = model_new.fit_generator(train, steps_per_epoch=train.n//batch_size, epochs=50, validation_data=test, validation_steps=5) # Learning		
# 9. Save trained model		
from keras.models import load_model		
model_new.save('cat_dog.h5') # Make pre-trained model and save		
files.download('cat_dog.h5')		
Epoch 1/50		

```
12/12 [=====] - 39s 2s/step - loss:  
2.2840 - accuracy: 0.4443 - val_loss: 3.0750 - val_accuracy:  
0.0000e+00  
Epoch 2/50  
12/12 [=====] - 11s 889ms/step - loss:  
1.2245 - accuracy: 0.4680 - val_loss: 0.5459 - val_accuracy:  
0.8400  
...  
...  
Epoch 49/50  
12/12 [=====] - 6s 468ms/step - loss:  
0.1389 - accuracy: 0.9101 - val_loss: 0.5759 - val_accuracy:  
0.8400  
Epoch 50/50  
12/12 [=====] - 6s 469ms/step - loss:  
0.1303 - accuracy: 0.9305 - val_loss: 1.0949 - val_accuracy:  
0.7200

import matplotlib.pyplot as plt
from PIL import Image
import numpy as np
import random
from keras.preprocessing.image import ImageDataGenerator
from keras import layers, callbacks, optimizers, models
from sklearn.metrics import classification_report
import sklearn

# 10. Import test data
shape=48
test = ImageDataGenerator(rescale=1./255).flow_from_directory(
    '/content/drive/MyDrive/archive/test_set', target_size=(shape,
    shape), color_mode = 'rgb', batch_size=5, shuffle = False, c
lass_mode='categorical') # Import test data from test folder

# 11. Import pre-trained model
from keras.models import load_model
model_import = load_model('cat_dog.h5') # Model import
result = model_import.predict_generator(test, steps=10) # Doing test
result = np.round(result) # Round result

# 12. Print result
print(test.class_indices) # Print class
print(result) # Print result

# 13. Evaluate model
```

```

    , score = model_import.evaluate(test, batch_size=32, verbose=0) # Calculate Accuracy
    score = score * 100.0
    print('Accuracy : %.3f' % (score)) # Print Accuracy

{'cats': 0, 'dogs': 1}
[[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 ...
 [1. 0.]
 [0. 1.]]
Accuracy : 65.461

x, y = test.next() # Split test to image and label
# 14. Print tested data
plt.figure(figsize=(15,15)) # Predict size of print
for i in range(5):
    plt.subplot(1,5,i+1) # Print 1*5 data
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    plt.imshow(x[i], cmap=plt.cm.binary) # Print image
    plt.xlabel("real : " + str(y[i]) + "\npredict : " + str(result[i])) # Print label
    plt.show() # Print conclusion

```



8.3. VIDEO PROCESSING

8.3.1. Video Processing for Object Detection

Whether it's a live stream on a personal blog or a security camera in a manufacturing facility, video data is a common commodity used daily. A machine learning appliance is quickly becoming a standard tool for video processing in a range of applications. In a word, video processing is a set of operations performed on each frame. Each frame contains decoding, calculation, and encoding activities. We may

generally assume that successive frames in a video are connected concerning their semantic contents using video classification, which is more than simply simple picture classification.

Although object recognition technology is occasionally utilized for stationary items, moving things is more commonly employed. Object identification utilizing these films is used in Tesla vehicles' autopilot, as well as a camera gazing at a factory's conveyor line. This real-time processing technology, made possible by the advancement of contemporary computer technology, aids in the faster recognition of the environment and the use of more appropriate techniques. As a result, this chapter aims to apply the h5 (model) file that was learned in section 8.2.3 (object detection) to video processing.

8.3.1.1. Python Implementation

```
1. Technology used
• Transfer learning
• Pretrained data

2. Reference
• https://whiteduck.tistory.com/160

3. Description
a) Download video
b) Video view
c) Import libraries
d) Import google drive
e) Import pre-trained model from the main folder of Google Drive
f) Preparing dataset and output
g) Predict
h) Download result

# 1. Download video https://whiteduck.tistory.com/160
!pip install youtube-dl # Install youtube-
dl for getting video from youtube
!youtube-dl https://www.youtube.com/watch?v=WDlu1OhvYBM -f 22 -o download.mp4 # Get video
```

```
Collecting youtube-dl
  Downloading
https://files.pythonhosted.org/packages/a4/43/1f586e49e68f8b41
c4be416302bf96ddd5040b0e744b5902d51063795eb9/youtube_dl-
2021.6.6-py2.py3-none-any.whl (1.9MB)
|████████████████████████████████████████████████████████| 1.9MB 7.5MB/s
Installing collected packages: youtube-dl
Successfully installed youtube-dl-2021.6.6
[youtube] WDlu1OhvYBM: Downloading webpage
[download] Destination: download.mp4
[download] 100% of 55.78MiB in 00:01

!ffmpeg -i download.mp4 -t 00:00:20 -c:v copy data.mp4 # Cut
video to 20s

# 2. Video view
from IPython.display import HTML
from base64 import b64encode
mp4 = open('data.mp4','rb').read()
video_url = "data:video/mp4;base64," + b64encode(mp4).decode()

HTML("""
<video width=500 controls>
  <source src="%s" type="video/mp4">
</video>
"""\ % video_url)
```



```

# 3. Import libraries
from keras.models import load_model
import cv2
from google.colab.patches import cv2_imshow

# 4. Import google drive
from google.colab import drive
drive.mount('/content/drive')

# 5. Import pre-trained model from main folder of Google Drive
model_import = load_model('/content/drive/MyDrive/archive/cat_
dog.h5')

# 6. Preparing dataset and output
cap = cv2.VideoCapture('data.mp4') # Capture frames of data.m
p4
width_org = cap.get(cv2.CAP_PROP_FRAME_WIDTH) # Get width size
of data.mp4
height_org = cap.get(cv2.CAP_PROP_FRAME_HEIGHT) # Get height s
ize of data.mp4
fourcc_org = cv2.VideoWriter_fourcc(*'XVID')
writer = cv2.VideoWriter('result.mp4', fourcc_org, 60, (int(wi
dth_org), int(height_org))) # Prepareing result video

# 7. Predict
while(cap.isOpened()):
    success, img = cap.read() # Get data from captured set
    if success == False: break # If there is no left data(end o
f the video), break
    img_test = cv2.resize(img, dsize = (48, 48)) # Decoding ima
ge to 48*48
    img_test = img_test.reshape(1, 48, 48, -
1) # Make shape for 2D CNN network
    result = model_import.predict_generator(img_test, steps=10)
# Predict
    print(result)
    if round(float(result[0, 0])) == 1: # If cat
        cv2.putText(img, "CAT", (10, 50), cv2.FONT_HERSHEY_SIMPLEX
, 0.5, (0,0,0), 1, cv2.LINE_AA) # Print "CAT" at 10, 50 of vid
eo
    elif round(float(result[0, 1])) == 1: #If dog
        cv2.putText(img, "DOG", (10, 50), cv2.FONT_HERSHEY_SIMPLEX
, 0.5, (0,0,0), 1, cv2.LINE_AA) # Print "DOG" at 10, 50 of vid
eo

    writer.write(img) # Put result image to result video's frame

```

```
cv2.imshow(img) # Print result image  
cv2.destroyAllWindows()
```



```
# 8. Download result  
from google.colab import files  
files.download('result.mp4')
```

8.4. MEDICAL IMAGING

Medical imaging refers to the techniques and procedures used to produce pictures of various human body areas for diagnostic and therapeutic reasons in digital health. Medical imaging refers to a variety of radiological imaging methods. All diagnostic and therapeutic investigations/interventions performed in a conventional radiology department are classified as medical imaging. It includes various imaging techniques and procedures to scan the human body for diagnostic, therapeutic and follow-up reasons. It plays a significant part in public health programs aimed at all demographic groups.

Imaging is an excellent resource for many diseases and a valuable tool for physical therapists when appropriately utilized. It is critical to understand when imaging is essential, as superfluous imaging wastes money and increases the risk of early surgery. Medical imaging encompasses a wide range of radiological imaging methods, including:

1. X-ray
2. Fluoroscopy
3. Magnetic resonance imaging (MRI)
4. Magnetic resonance imaging (fMRI)
5. Ultrasonography (often known as ultrasound)
6. Endoscopy
7. Positron emission tomography (PET)

8.4.1. Neuroimaging Application With fMRI

Machine learning-based vision recognition has gone a long way, as we have seen. Its performance is already exceeding human cognitive capacity. Therefore scientists are considering how to put this impressive capability to use. This has brought the bio and medical areas into the limelight. Even for major illnesses, there are situations where only minor alterations emerge in this field. If people label it, there is a chance it will be misdiagnosed. On the other hand, machine learning is predicted to lower the likelihood of misdiagnosis and lessen the burden on patients and society by responding sensitively to relatively tiny changes and accurately classifying them. As a result, this chapter aims to divide it into three groups using fMRI brain scans.

The tiny variations in blood flow that occur with brain activity are measured using functional magnetic resonance imaging (fMRI). It can be used to look at the brain's functional architecture, assess the consequences of a stroke or other disease, or help with brain therapy. Other imaging techniques may not be able to identify problems in the brain that fMRI can.

For learning how a normal, sick, or damaged brain works and determining the possible dangers of surgery or other invasive brain therapies, fMRI is becoming the diagnostic tool of choice. During an fMRI examination, you will be asked to do things like tapping your fingers or toes, pursing your lips, moving your tongue, reading, seeing images, listening to speech, and/or playing simple word games. Increased metabolic activity in the brain regions responsible for these activities will result as a result of this. This activity, which includes the dilation of blood vessels, chemical changes, and more oxygen transport, may then be seen on MRI scans.

8.4.1.1. Python Implementation

```
1. Technology used
• SVM
• fMRI

2. Reference
• https://github.com/poldrack/fmri-classification-example

3. Description
a) Download dataset from github
b) Install nibabel and nilearn
c) Install keras-utils
d) Import libraries
e) Load dataset
f) Make labels
g) print dates
h) Learning
i) Shuffle labels and test again

# 1. Download dataset from github
!git clone https://github.com/poldrack/fmri-classification-
example.git

# 2. Install nibabel and nilearn
!pip install nibabel nilearn

# 3. Install keras-utils
!pip install keras-utils

# 4. Import libraries
import os
import nibabel, numpy
import sklearn.svm
import nilearn.input_data
import nilearn.plotting
from tensorflow.keras.utils import to_categorical

%matplotlib inline

# 5. Load dataset
a=nilearn.input_data.NiftiMasker(mask_img='fmri-
classification-example/nback_mask.nii.gz')
```

```

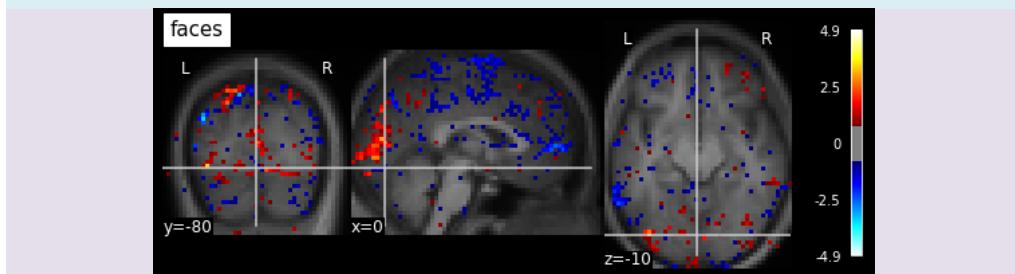
dataset=a.fit_transform('fmri-classification-
example/nback_zstats1-11-21_all.nii.gz')

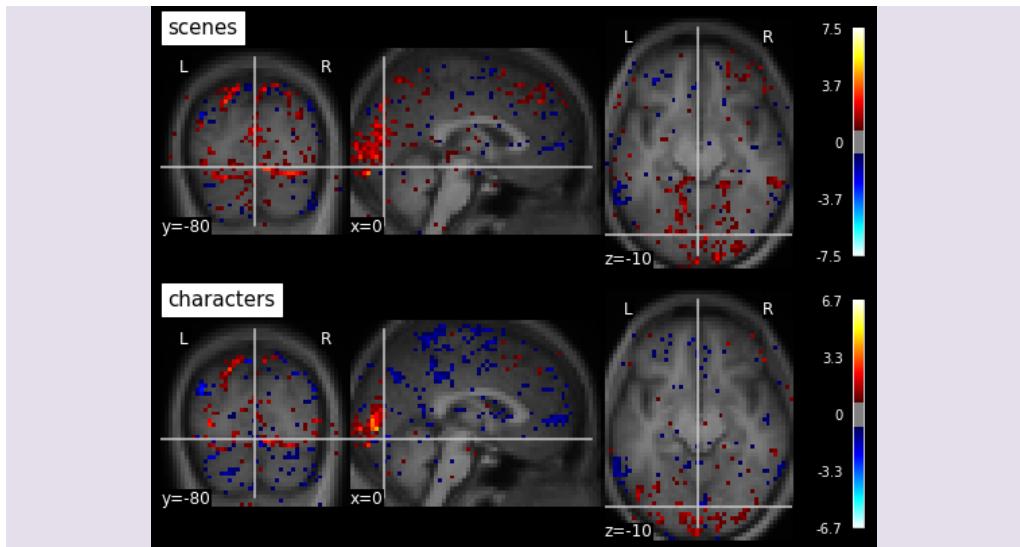
# 6. Make labels
label=numpy.zeros(dataset.shape[0]) # Make numpy array that al
l 0
label[15:30]=1 # From 15 to 30, change 0 to 1
label[30:]=2 # From 30, change 0 to 2

indicator=numpy.kron(numpy.ones(3),numpy.arange(15)) # Make se
ssion indicator

# 7. Print data
thresh=0.8
coords=(0,-80,-10) # Set coords of line
background=nibabel.load('fmri-classification-
example/TRIO_Y_NDC_333_fsl.nii.gz') # Import background image
face=nibabel.load('fmri-classification-
example/nback_zstat1_mean.nii.gz') # Import faces image
nilearn.plotting.plot_stat_map(face, bg_img=background, threshol
d=thresh, title='faces', cut_coords=coords) # Print faces image
scene=nibabel.load('fmri-classification-
example/nback_zstat11_mean.nii.gz') # Import scenes image
nilearn.plotting.plot_stat_map(scene, bg_img=background, thresho
ld=thresh, title='scenes', cut_coords=coords) # Print scenes im
age
chars=nibabel.load('fmri-classification-
example/nback_zstat21_mean.nii.gz') # import characters image
nilearn.plotting.plot_stat_map(chars, bg_img=background, thresho
ld=thresh, title='characters', cut_coords=coords) # Print charac
ters image

```





```
# 8. Learning
def run_classifier(shuffle_labels=False, verbose=True): # Make
    run_classifier function
    accuracy=numpy.zeros((15,3)) # Make acc array for import d
    ata

    for i in range(15):
        obs_tr=indicator!=i # Define train set
        obs_te=indicator==i # Define test set
        trainX=dataset[obs_tr,:] # Make train data
        testX=dataset[obs_te,:] # Make test data
        trainY=label[obs_tr] # Make train labels
        if shuffle_labels:
            numpy.random.shuffle(trainY) # Shuffle train label
        s
        testY=label[obs_te] # Make test labels
        model=sklearn.svm.SVC(kernel='linear') # Make SVM mod
    el
        model.fit(trainX,trainY) # Training
        p=model.predict(testX) # Test
        accuracy[i,:]=p==testY # Calculate test accuracy
        acc_tr=model.predict(trainX)==trainY # Calculate train
    accuracy
        if verbose:
            print('Session number %d'%i) # Print session numbe
    r
```

```
        print('Training acc:',numpy.mean(acc_tr)) # Print
training accuracy
        print('Test acc:',numpy.mean(accuracy[i,:])) # Pr
int test accuracy
    return accuracy

accr=run_classifier() # Using function
acc_m=numpy.mean(accr,0) # Calculate mean accuracy
print('')
print('Mean acc: %0.3f'%numpy.mean(accr)) # Print average accu
racy
print('Faces acc: %0.3f'%acc_m[0]) # Print faces average accu
racy
print('Scenes acc: %0.3f'%acc_m[1]) # Print Scenes average acc
uracy
print('Characters acc: %0.3f'%acc_m[2]) # Print characters ave
rage accuracy

Session number 0
Training acc: 1.0
Test acc: 0.3333333333333333
Session number 1
Training acc: 1.0
Test acc: 1.0
Session number 2
Training acc: 1.0
Test acc: 0.6666666666666666
Session number 3
Training acc: 1.0
Test acc: 0.6666666666666666
...
...
Session number 14
Training acc: 1.0
Test acc: 0.6666666666666666

Mean acc: 0.667
Faces acc: 0.667
Scenes acc: 0.733
Characters acc: 0.600

# 9. Shuffle labels and test again
accr=run_classifier(shuffle_labels=True,verbose=False) # Test
acc_m=numpy.mean(accr,0) # Calculate mean accuracy
print('Shuffled')
```

```
print('Mean acc: %0.3f'%numpy.mean(acc)) # Print average accuracy
print('Faces acc: %0.3f'%acc_m[0]) # Print faces average accuracy
print('Scenes acc: %0.3f'%acc_m[1]) # Print Scenes average accuracy
print('Characters acc: %0.3f'%acc_m[2]) # Print characters average accuracy

Shuffled
Mean acc: 0.356
Faces acc: 0.333
Scenes acc: 0.467
Characters acc: 0.267
```

8.4.2. Tumor Detection Using MRI

The body's hydrogen atoms are mapped using magnetic resonance imaging (MRI). Because hydrogen atoms have a single proton and a high magnetic moment, they are excellent for MRI. An MRI scanner is a giant, strong magnet in which the patient is placed in its most basic form. The magnetic field generated by the magnets causes each proton in the hydrogen atom to resonate, allowing the machine to determine the proton's location. Because water molecules make up around 75 percent of human bodies, MR imaging can acquire accurate and detailed pictures of the examined body area.

MRI is the preferred diagnostic technique for neurological cancers because it is more sensitive than CT for tiny tumors and allows for better imaging of the posterior fossa. Many central nervous system disorders, such as demyelinating diseases, schizophrenia, dementia, cerebrovascular disease, viral diseases, and epilepsy, benefit from the contrast given by grey and white matter.

Recent machine learning models show great discriminating capacity even with minor changes as mentioned in the previous section. Cancer, one of humanity's most feared diseases, is no exception. Early cancer is frequently misdiagnosed as a stomach ulcer or a variety of tumors, making it a complex condition to identify. As a result, machine learning is gaining traction as an area in which it is necessary. As a result, this chapter aims to develop a model for utilizing CNN to diagnose the presence or absence of cancer.

8.4.2.1. Python Implementation

1. Technology used

- CNN
- Medical imaging

2. Reference

- <https://medium.com/hyunjulie/%EC%BA%90%EA%B8%80%EA%B3%BC-%EA%B5%AC%EA%B8%80-colab-%EC%97%B0%EA%B2%B0%ED%95%B4%EC%A3%BC%EA%B8%B0-6a274f6de81d>
- <https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection>
- <https://www.kaggle.com/ruslank1/brain-tumor-detection-v1-0-cnn-vgg-16/data>
- <https://www.tensorflow.org/tutorials/keras/classification>

3. Description

- a) Preparing Kaggle
- b) Dataset download from kaggle
- c) Import libraries
- d) Split data
- e) Import dataset
- f) print dates
- g) Make CNN model
- h) Learning
- i) Test
- j) Print result
- k) Evaluate model
- l) Print tested data

```
# 1. Preparing Kaggle      https://medium.com/hyunjulie/%EC%BA%90%EA%B8%80%EA%B3%BC-%EA%B5%AC%EA%B8%80-colab-%EC%97%B0%EA%B2%B0%ED%95%B4%EC%A3%BC%EA%B8%B0-6a274f6de81d

# Kaggle install
!pip install kaggle

# Upload json file
from google.colab import files
files.upload()

# Json file import
!mkdir -p ~/.kaggle
```

```

!cp kaggle.json ~/.kaggle/
# Fixing permission
!chmod 600 ~/.kaggle/kaggle.json

# 2. Dataset download from kaggle https://www.kaggle.com/navoneel/brain-mri-images-for-brain-tumor-detection
!kaggle datasets download -d navoneel/brain-mri-images-for-brain-tumor-detection
!unzip brain-mri-images-for-brain-tumor-detection.zip

# 3. Import libraries
import os
import shutil
from keras.preprocessing.image import ImageDataGenerator
from keras.layers.convolutional import Conv2D, MaxPooling2D
from keras.models import Sequential
from keras.layers import Dense, Flatten, Dropout
from pathlib import Path
import matplotlib.pyplot as plt
import numpy as np

path = './brain_tumor_dataset/'
# 4. Split data # https://www.kaggle.com/ruslankl/brain-tumor-detection-v1-0-cnn-vgg-16/data
for data in os.listdir(path): # For something in path
    if not data.startswith('.'): # If "data" is not empty
        num = len(os.listdir(path + data)) # num = file counts of folder(path + data)
        Path("./test/" + data.upper()).mkdir(parents=True, exist_ok=True) # Make "test" directory
        Path("./train/" + data.upper()).mkdir(parents=True, exist_ok=True) # Make "train" directory
        Path("./validation/" + data.upper()).mkdir(parents=True, exist_ok=True) # Make "validation" directory
        for (n, name) in enumerate(os.listdir(path + data)):
            # For something in folder(path + data)
            image = path + data + '/' + name # Make list of image data
            if n < 5: # Up to 50%
                shutil.copy(image, 'test/' + data.upper() + '/' + name) # Copy files to "test" directory
            elif n < 0.8*num: # Up to 80%
                shutil.copy(image, 'train/' + data.upper() + '/' + name) # Copy files to "train" directory

```

```

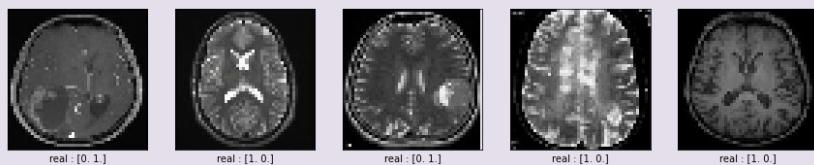
        else:
            shutil.copy(image, 'validation/' + data.upper()
+ ' / ' + name) # Copy files to "validation" directory

# 5. Import dataset
shape=48
train = ImageDataGenerator(rescale=1./255).flow_from_directory(
    './train', target_size=(shape, shape), batch_size=5, color_mode = 'rgb', class_mode='categorical') # Import train data from train folder
val = ImageDataGenerator(rescale=1./255).flow_from_directory(
    './validation', target_size=(shape, shape), color_mode = 'rgb',
, batch_size=5, shuffle = False, class_mode='categorical') # Import validation data from validation folder
test = ImageDataGenerator(rescale=1./255).flow_from_directory(
    './test', target_size=(shape, shape), color_mode = 'rgb', batch_size=5, shuffle = False, class_mode='categorical') # Import test data from test folder

Found 193 images belonging to 2 classes.
Found 50 images belonging to 2 classes.
Found 10 images belonging to 2 classes.

x, y = train.next() # Split train to image and label
# 6. Print data      https://www.tensorflow.org/tutorials/keras/classification
plt.figure(figsize=(15,15)) # Predict size of print
for i in range(5):
    plt.subplot(1,5,i+1) # Print 1*5 data
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    plt.imshow(x[i], cmap=plt.cm.binary) # Print image
    plt.xlabel("real : " + str(y[i])) # Print label
plt.show() # Print conclusion

```



```
# 7. Make CNN model
model = Sequential() # Start making model
model.add(Conv2D(filters = 64, kernel_size = 3, activation = 'relu', input_shape = (shape, shape, 3))) # 2D CNN layer with input layer
model.add(Conv2D(filters=64, kernel_size=3, activation='relu')) # 2D CNN layer
model.add(Dropout(0.5)) # 50% Dropout
model.add(Flatten()) # Reduce dimension
model.add(Dense(100, activation='relu')) # Fully connected layer
model.add(Dense(2, activation='softmax'))# Fully connected layer with output layer
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy']) # Compile model
model.summary() # Check model
```

Model: "sequential"

Layer (type)	Output Shape	Param #
<hr/>		
=		
conv2d (Conv2D)	(None, 46, 46, 64)	1792
<hr/>		
conv2d_1 (Conv2D)	(None, 44, 44, 64)	36928
<hr/>		
dropout (Dropout)	(None, 44, 44, 64)	0
<hr/>		
flatten (Flatten)	(None, 123904)	0
<hr/>		
dense (Dense)	(None, 100)	12390500
<hr/>		
dense_1 (Dense)	(None, 2)	202
<hr/>		
=		
Total params: 12,429,422		
Trainable params: 12,429,422		
Non-trainable params: 0		

```
# 8. Learning
hist = model.fit_generator(train, steps_per_epoch=train.n//32,
    epochs=50, validation_data=val, validation_steps=5) # Learning

Epoch 1/50
6/6 [=====] - 18s 404ms/step - loss: 1.9524 - accuracy: 0.5476 - val_loss: 0.9589 - val_accuracy: 0.2400
Epoch 2/50
6/6 [=====] - 2s 286ms/step - loss: 0.7243 - accuracy: 0.5662 - val_loss: 0.5409 - val_accuracy: 0.7200
Epoch 3/50
6/6 [=====] - 2s 286ms/step - loss: 0.5018 - accuracy: 0.7933 - val_loss: 1.6130 - val_accuracy: 0.3200
Epoch 4/50
6/6 [=====] - 2s 282ms/step - loss: 0.5439 - accuracy: 0.7452 - val_loss: 0.5569 - val_accuracy: 0.6400
...
...
Epoch 50/50
6/6 [=====] - 2s 287ms/step - loss: 5.7654e-04 - accuracy: 1.0000 - val_loss: 1.3477 - val_accuracy: 0.8000

# 9. Test
result = model.predict_generator(test, steps=10) # Doing test
result = np.around(result) # Round result

# 10. Print result
print(test.class_indices) # Print class
print(result) # Print result

# 11. Evaluate model
_, score = model.evaluate(test, batch_size=32, verbose=0) # Calculate Accuracy
score = score * 100.0
print('Accuracy : %.3f' % (score)) # Print Accuracy

{'NO': 0, 'YES': 1}
[[0. 1.]
```

```

[1. 0.]
[1. 0.]
[1. 0.]
[1. 0.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
[0. 1.]
Accuracy : 90.000

x, y = test.next() # Split test to image and label
# 12. Print tested data
plt.figure(figsize=(15,15)) # Predict size of print
for i in range(5):
    plt.subplot(1,5,i+1) # Print 1*5 data
    plt.xticks([]) # Not using x-axis graduation
    plt.yticks([]) # Not using y-axis graduation
    plt.grid(False) # Not using grid
    plt.imshow(x[i], cmap=plt.cm.binary) # Print image
    plt.xlabel("real : " + str(y[i]) + "\npredict : " + str(result[i])) # Print label
    plt.show() # Print conclusion

```

8.5. COMPUTATIONAL LINGUISTIC

Computational linguistics is a scientific and technical subject that focuses on comprehending written and spoken language from a computational standpoint and developing artifacts that can process and generate language in bulk or a conversational context. A computational knowledge of language gives insight into thinking and intelligence to the degree that language is a mirror of thought. Language is our most natural and versatile mode of communication. Linguistically competent computers would considerably simplify our interactions with machines.

and software of all kinds, putting the enormous textual and other information at our fingertips in ways that fit our requirements.

The field's practical objectives are numerous and diverse. Some of the most notable are: efficient text retrieval on a specific topic; effective machine translation; question answering, which ranges from simple factual questions to those requiring inference and descriptive or discursive answers; text summarization; text or spoken language analysis for a topic, sentiment, or other psychological attributes; dialogue amplification; and text summarization.

Computational linguistics is concerned with creating and studying methods that make these and other similar applications possible. As a result, the analysis might focus on primary language difficulties like modeling the meaning of words and identifying the grammatical structure of sentences to more sophisticated applications like machine translation or fact-checking claims. Statistical and computational techniques such as neural networks and logic-based procedures are used in the analysis. Computational linguistics contributes to the advancement of artificial intelligence and catalyzes new ideas in the area.

8.5.1. Sentiment Analysis Using NLP

Natural language processing (NLP) is a field of computer science, and more specifically, a branch of artificial intelligence (AI), concerned with teaching computers to comprehend text and spoken words in the same manner that humans do.

NLP integrates statistical, machine learning, and deep learning models with computational linguistics rule-based modeling of human language. These technologies, when used together, allow computers to interpret human language in the form of text or speech data and 'understand' its whole meaning, including the speaker or writer's purpose and mood.

Sentiment analysis is a technique for determining whether data is good, harmful, or neutral using natural language processing. Sentiment analysis is frequently used on textual data to assist organizations in tracking brand and product sentiment in consumer feedback and better understanding customer requirements. A sentiment analysis system for text analysis combines natural language processing and machine learning approaches to give weighted sentiment ratings to entities, topics, themes, and categories inside a sentence or phrase.

Sentiment analysis is a sophisticated text analysis tool that uses machine learning and deep learning algorithms to mine unstructured data for opinion and emotion automatically.

Deep learning is seen as the next step in the evolution of machine learning. It is an artificial neural network that connects algorithms to replicate how the human brain functions. It has enabled numerous practical applications of machine learning, such as customer support automation and self-driving automobiles. Let us take a deeper look at how deep learning may help with sentiment analysis.

8.5.1.1. Python Implementation

```
1. Technology used
• LSTM
• NLP

2. Reference
• https://www.kaggle.com/ngypt/lstm-sentiment-analysis-keras?select=database.sqlite
• https://wikidocs.net/24586

3. Description
a) Import libraries
b) Download dataset
c) Reduce the length of review to 300
d) Reduce review to 300
e) Print data
f) Make LSTM model
g) Learning
h) Test
i) Define test accuracy

# 1. Import libraries
import numpy as np
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM
from keras.datasets import imdb

# 2. Download dataset
```

```
(trainX, trainY), (testX, testY) = imdb.load_data(num_words =  
5000) # Load and split dataset(5,000 popular words only)  
  
# Set data labels  
data_labels = ['Negative', 'Positive']  
  
# 3. Reduce length of review to 300  
max_len = 300 # Set max length of each review  
trainX = pad_sequences(trainX, maxlen=max_len) # set length t  
o train data  
testX = pad_sequences(testX, maxlen=max_len) # set length to  
test data  
  
# 4. Reduce review to 300  
trainX = trainX[:300, :]  
trainY = trainY[:300]  
testX = testX[:300, :]  
testY = testY[:300]  
  
# 5. Print data  
index = imdb.get_word_index() # Import indexset  
index2={}  
for key, value in index.items():  
    index2[value+3] = key # Change value(num) to sentence  
for index, token in enumerate(("<pad>", "<sos>", "<unk>")):  
    index2[index]=token  
print(' '.join([index2[index] for index in trainX[0]])) # Prin  
t data  
  
dim = 128 # Set embedding dimension  
output = 196 # Set LSTM output size  
  
# 6. Make LSTM model  
model = Sequential() # Make start  
model.add(Embedding(5000, dim, input_length = trainX.shape[0]))  
    # Add Embedding  
model.add(LSTM(output, dropout=0.2, recurrent_dropout=0.2)) #  
Add LSTM  
model.add(Dense(2, activation='softmax')) # Add output layer  
model.compile(loss = 'sparse_categorical_crossentropy', optimi  
zer='adam', metrics = ['accuracy']) # Model compile  
model.summary() # Print model
```

```

Model: "sequential"
-----
Layer (type)          Output Shape         Param #
=====
embedding (Embedding) (None, 300, 128)      640000
lstm (LSTM)           (None, 196)          254800
dense (Dense)         (None, 2)            394
=====
Total params: 895,194
Trainable params: 895,194
Non-trainable params: 0
-----

# 7. Learning
batch_size = 32
model.fit(trainX, trainY, epochs = 50, batch_size=batch_size,
verbose = 1) # Training

Epoch 1/50
10/10 [=====] - 34s 1s/step - loss: 0.6945 - accuracy: 0.4781
Epoch 2/50
10/10 [=====] - 11s 1s/step - loss: 0.6820 - accuracy: 0.5059
Epoch 3/50
10/10 [=====] - 11s 1s/step - loss: 0.6163 - accuracy: 0.6216
...
...
Epoch 50/50
10/10 [=====] - 11s 1s/step - loss: 5.7063e-05 - accuracy: 1.0000

# 8. Test
result = model.predict_generator((testX, testY), steps=10) # Doing test

```

```
result = np.round(result) # Round result
for i in range(len(result)):
    print('number : ', i) # Print data number
    print("real : ", testY[i], data_labels[testY[i]]) # Print real index
    print("predict : ", np.argmax(result[i]), data_labels[np.argmax(result[i])]) # Print predicted index
    print('\n\n')

# 9. Define test accuracy
_, score = model.evaluate(testX, testY, batch_size=32, verbose=0)
score = score * 100.0
print('# test accuracy: %.3f' % (score))

number : 0
real : 0 Negative
predict : 1 Positive

number : 1
real : 1 Positive
predict : 1 Positive

number : 2
real : 1 Positive
predict : 1 Positive

number : 3
real : 0 Negative
predict : 0 Negative
...
...
number : 298
real : 1 Positive
predict : 0 Negative

number : 299
real : 0 Negative
predict : 0 Negative

# test accuracy: 58.000
```

8.6. LIST OF POSSIBLE REAL-WORLD APPLICATIONS

Our lives are now more pleasant and preferred than they were previously, thanks to the mysterious touch of science. Science's significance in our everyday lives is evident. Science's impact on our lives cannot be overlooked or ignored. If we attempt to comprehend the precise impact of science on our lives, we will discover that these are the results of employing artificial intelligence and machine learning applications. In this part, we attempt to capture the outstanding real-time machine learning applications that will transform our view of life.

8.6.1. Image Recognition

In the real world, image recognition is a well-known and often used example of machine learning. The intensity of the pixels in black and white or color pictures may recognize an item as a digital image.

8.6.2. Sentiment Analysis

Another real-time machine learning application is sentiment analysis. Opinion mining, sentiment categorization, and other terms are also used.

8.6.3. Speech Recognition

The speech-to-text translation is possible using machine learning. Live voice and recorded speech may both be converted to text files using specific software programs.

8.6.4. Medical Diagnosis

Disease diagnosis can be aided by machine learning.

8.6.5. News Classification

Another benchmark use of a machine learning technique is news categorization.

8.6.6. Predictive Analysis

Machine learning may divide accessible data into categories, which are subsequently defined by analyst-specified rules. The analysts can determine the likelihood of a defect after the categorization is complete.

8.6.7. Video Surveillance

Text documents and other media files, such as music and photos, contain less information than a tiny video file. As a result, collecting valuable information from video, *i.e.*, an automated video surveillance system, has become a hot topic of research.

8.6.8. Text Extraction

Machine learning can parse unstructured data and extract structured information. Customers provide massive amounts of data to businesses. The process of annotating datasets for predictive analytics tools is automated using a machine learning algorithm.

8.6.9. Email Categorization

A machine learning technique is used to categorize emails and filter spam automatically.

8.6.10. Social Media Analysis

Social media uses machine learning to produce appealing and beautiful services for its users, such as individuals you may know, recommendations, and response choices.

CONCLUDING REMARKS

We have gone through every facet of machine learning and deep learning models so far, and we have a good understanding of their underlying architecture. The readers were introduced to state-of-the-art, real-world applications of machine learning and deep learning algorithms in this chapter. The chapter has explored real-world applications from every corner of recent developments, from the everyday use of facial recognition to object detection. Not only that, but this chapter also covered how machine learning and deep learning are used in everyday life. This chapter has addressed the most critical applications in the domains of pattern recognition, video processing, medical imaging, and computational linguistics, among others. The Python implementation of all of the applications was provided in this chapter. This chapter also discussed several additional essential real-world applications that we utilize regularly.

CHAPTER 9

Conclusions

All of the chapters in this book are devoted to machine learning and deep learning architecture. As the name implies, it focuses on real-world machine learning applications and deep learning techniques. The following is a general outline of the book:

The introductory chapter introduced the fundamental concepts of artificial intelligence and the gradual growth of artificial intelligence, giving rise to machine learning and deep learning. It threw deep insights into the need for learning and the concept of learning in the machine learning framework. This chapter also stated the importance of AI and its various dimensions. In the end, this chapter introduces machine learning and its variants. It also shortly described each type of machine learning algorithm.

The second chapter introduced supervised machine learning algorithms. This chapter explained the algorithms such as Decision Trees, Random Forests, K-Neighbors, Name Bias Classifiers, and Support Vector Machines. Each algorithm begins with an overview, then is described with an algorithm framework and hands-on examples. A detailed program for Python is provided at the end of each algorithm to understand a practical understanding of the functional behavior of the classification. The Python code runs on real data sets and ultimately gives the reader an in-depth knowledge of algorithm applications.

The chapter on clustering algorithms was introduced to the readers as a part of unsupervised machine learning algorithms. This chapter described the state-of-the-art clustering algorithms. This chapter presented an elaborative definition of k-mean clustering, hierarchical clustering, and self-organizing map. It also defined the algorithmic framework of each algorithm with hands-on examples with detailed Python codes and outputs.

Next, the readers were given a thorough understanding of regression analysis in the introductory chapter. The notion of regression is used in both statistics and computer science, particularly in machine learning. The principle, on the other hand, hasn't changed, although the applications have. The two most used regression analysis techniques, linear and logistic regression, were covered in this chapter. Each algorithm is discussed in full here, along with examples on how to use them.

In addition, we learned more in-depth about linear and logistic regression by showing them with a Python application on real-world data.

In the final part of section II, reinforcement learning (RL) was introduced. It a machine learning subfield similar to supervised or unsupervised learning but differs in numerous aspects. It is more difficult to apply to real-world business scenarios since it requires simulated data and surroundings. On the other hand, RL technology is promising since its learning process is inherent to sequential decision-making settings. This chapter clearly explained the algorithm and its usage.

The chapter on deep learning introduced the reader to the most up-to-date, state-of-the-art deep learning algorithms in this chapter. Deep learning is a contemporary discipline of machine learning capable of recognizing the nature of data and comprehending the underlying patterns in it on its own. This chapter covered everything from neural network structure to advanced neural networks like convolutional neural networks and recurrent neural networks. It covered artificial neural networks, feed-forward neural networks, convolutional neural networks, and recurrent neural networks, with key topics like backpropagation, gradient descent, activation functions, and optimizations highlighted. This chapter will undoubtedly improve the readers' knowledge of advanced technologies by providing hands-on examples and a Pythonic approach to real-world applications.

The chapter on feature selection approaches covered the most current feature selection strategies used in conjunction with machine learning algorithms. The selection of features is an essential factor in improving the performance of any machine learning system. The two types of feature selection algorithms discussed in this chapter were filter-based and evolutionary-based. This chapter discusses two types of filter-based algorithms: hypothetical testing, such as the t-test, z-test, ANOVA, MANOVA, and correlation-based, such as Pearson's correlation, Chi-square test, and Spearman's rank correlation. This chapter solely covered evolutionary methods, including genetic algorithms, particle swarm optimization, and ant colony optimization. This chapter goes through each of the methods in-depth and the optimized algorithm for conducting the feature selection technique.

We have gone through every facet of machine learning and deep learning models so far, and we have a good understanding of their underlying architecture. The readers were introduced to state-of-the-art real-world applications of machine learning and deep learning algorithms in this chapter. The chapter has explored real-world applications from every corner of recent developments, from the everyday use of facial recognition to object detection. Not only that, but this chapter

also covered how machine learning and deep learning are used in everyday life. This chapter has addressed the most critical applications in the domains of pattern recognition, video processing, medical imaging, and computational linguistics, among others. The Python implementation of all of the applications was provided in this chapter. This chapter also discussed several additional essential real-world applications that we utilize regularly.

REFERENCES

- [1] S. Russell, and P. Norvig, *Artificial intelligence: a modern approach.*, 3rd ed Pearson, 2011.
- [2] "Understanding the Four Types of Artificial Intelligence", Apr. 23, 2021. [Online]. Available: 2021. <https://www.govtech.com/computing/understanding-the-four-types-of-artificial-intelligence.html> [Accessed: 7th Jul. 2021].
- [3] O. Theobald, *Machine Learning For Absolute Beginners: A Plain English Introduction (Second Edition) (Machine Learning From Scratch Book 1)*, 2nd ed Scatterplot Press, 2017.
- [4] M. Fenner, *Machine learning with Python for Everyone*. Addison-Wesley Professional, 2019.
- [5] O. Theobald, *Machine Learning For Absolute Beginners: A Plain English Introduction (Second Edition) (Machine Learning From Scratch Book 1)*, 2nd ed Scatterplot Press, 2017.
- [6] D. Conway, and J.M. White, *Machine Learning for Hackers: Case Studies and Algorithms to Get You Started.*, 1st ed O'Reilly, 2012.
- [7] J.P. Mueller, and L. Massaron, *Data Science Programming All-in-One For Dummies.*, 1st ed For Dummies, 2020.
- [8] "Linear Regression — ML Glossary documentation", Available: https://ml-cheatsheet.readthedocs.io/en/latest/linear_regression.html#making-predictions [Accessed: 5th Jun. 2021].
- [9] "Linear Regression in Machine learning – Javatpoint", Available: <https://www.javatpoint.com/linear-regression-in-machine-learning> [Accessed: 1st Jul, 2021].
- [10] J. Brownlee, *Linear Regression for Machine Learning*, 2021. Available: <https://machinelearningmastery.com/linear-regression-for-machine-learning/> [Accessed: 3rd Jul. 2021].
- [11] "Introduction to linear regression analysis", [Online]. Available: <https://people.duke.edu/%7Ernau/regintro.htm> [Accessed: 6th Jul. 2021].
- [12] A. Ye, *Markov decision process in reinforcement learning: everything you need to know.*, 2021. Available: <https://neptune.ai/blog/markov-decision-process-in-reinforcement-learning> [Accessed: 12th Jul. 2021].
- [13] A. Choudhary, *Nuts & Bolts of Reinforcement Learning: Model Based Planning using Dynamic Programming.*, 2021.<https://www.analyticsvidhya.com/blog/2018/09/reinforcement-learning-model-based-planning-dynamic-programming/> [Accessed: 13th Jul. 2021].
- [14] S. Paul, *An introduction to Q-Learning: Reinforcement Learning*, 2020. <https://blog.floydhub.com/an-introduction-to-q-learning-reinforcement-learning/> [Accessed: 16th Jul. 2021]
- [15] J. Howard, and S. Gugger, *Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD.*, 1st ed O'Reilly Media, 2019.
- [16] J. Le, "The 8 neural network architectures machine learning researchers need to learn", Available: <https://www.kdnuggets.com/2018/02/8-neural-network-architectures-machine-learning-researchers-need-learn.html> [Accessed: 19th Jul. 2021].
- [17] <https://developer.ibm.com/technologies/artificial-intelligence/articles/cc-machine-learning-deep-learning-architectures/> n.d. [Accessed: 20th Jul. 2021].
- [18] "Deep learning architectures", Available: <https://www.xenonstack.com/blog/artificial-neural-network-applications> [Accessed: 19th Jul. 2021].
- [19] "Everything you need to know about neural networks", Available: <https://hackernoon.com/everything-you-need-to-know-about-neural-networks-8988c3ee4491> [Accessed: 19th Jul. 2021].
- [20] "4.7. forward propagation, backward propagation, and computational graphs — dive into deep learning 0.16.6 documentation", Available: https://d2l.ai/chapter_multilayer-perceptrons/backprop.html [Accessed: 19th Jul. 2021].
- [21] S. Ruder, "An overview of gradient descent optimization algorithms", Available: <https://ruder.io/optimizing-gradient-descent/> [Accessed: 20th Jul. 2021].

- [22] "CS231n convolutional neural networks for visual recognition", [Online] <https://cs231n.github.io/convolutional-networks/> [Accessed: 19th Jul. 2021].
- [23] "Convolutional Neural Networks (CNN)", Available: <https://www.analyticsvidhya.com/blog/2021/05/convolutional-neural-networks-cnn/> [Accessed: 19th Jul. 2021].
- [24] "CS 230 - convolutional neural networks cheatsheet", Available: <https://stanford.edu/%7Eshervine/teaching/cs-230/cheatsheet-convolutional-neural-networks> [Accessed: 19th Jul. 2021].
- [25] "A Beginner's Guide to LSTMs and Recurrent Neural Network", Available: <https://wiki.pathmind.com/lstm> [Accessed: 19th Jul. 2021]
- [26] A. Zheng, and A. Casari, *Feature Engineering for Machine Learning: Principles and Techniques for Data Scientists.*, 1st ed O'Reilly Media, 2018.
- [27] S. Kaushik, *Introduction to Feature Selection methods with an example (or how to select the right variables?)*, 2020.<https://www.analyticsvidhya.com/blog/2016/12/introduction-to-feature-selection-methods-with-an-example-or-how-to-select-the-right-variables/> [Accessed: 19th Jul. 2021].
- [28] "Z-Test Definition", Available: <https://www.investopedia.com/terms/z/z-test.asp> [Accessed: 19th Jul. 2021].
- [29] S. Meena, *Statistics for analytics and data science: hypothesis testing and Z-test vs. T-Test*, 2020.<https://www.analyticsvidhya.com/blog/2020/06/statistics-analytics-hypothesis-testing-z-test-t-test/> [Accessed: 19th Jul. 2021].
- [30] M. Thakur, *Z-Test vs T-Test*, 2021. WallStreetMojo. Available: <https://www.wallstreetmojo.com/z-test-vs-t-test/> [Accessed: 19th Jul. 2021].
- [31] "ANOVA Test: Definition, Types, Examples", Jun. 19, 2021. [Online]. Available: <https://www.statisticshowto.com/probability-and-statistics/hypothesis-testing/anova/> [Accessed: 19th Jul. 2021].
- [32] MANOVA, Available: https://www.statisticssolutions.com/free-resources/directory-of-statistical-analyses/manova/?_cf_chl_jschl__tk__=pmd_16d7d4fa1acf6d3f445cd2769d8972412bcb64ea-1626714547-0-gqNtZGzNAg2jcnBszQki [Accessed: 19th Jul. 2021].
- [33] "Chi-Square Test for Feature Selection - Mathematical Explanation", Available: <https://www.geeksforgeeks.org/chi-square-test-for-feature-selection-mathematical-explanation/> [Accessed: 19th Jul. 2021].
- [34] "ML | Chi-square Test for feature selection", Available: <https://www.geeksforgeeks.org/ml-chi-square-test-for-feature-selection/> [Accessed: 19th Jul. 2021].
- [35] "Introduction to Ant Colony Optimization", Available: <https://www.geeksforgeeks.org/introduction-to-ant-colony-optimization/> [Accessed: 19th Jul. 2021].
- [36] T. Segaran, *Programming Collective Intelligence: Building Smart Web 2.0 Applications.*, 1st ed O'Reilly Media, 2007.
- [37] A. Géron, *Hands-on machine learning with scikit-learn, keras, and tensorflow: concepts, tools, and techniques to build intelligent systems.*, 2nd ed O'Reilly Media, 2021.

SUBJECT INDEX

A

Activation functions 182, 183, 184, 185, 190, 191, 199, 200, 201, 202, 203, 204, 205, 209, 210, 211, 212, 213, 214
sigmoidal 210
threshold 184, 185
utilized 209
Adaptive piecewise linear (APL) 214
Agglomerative clustering algorithm 94
Algorithm(s) 17, 19, 32, 35, 55, 71, 72, 75, 88, 139, 172, 218, 256, 275, 280, 282, 284, 289, 290, 291, 332, 333
filter-based 256, 289, 333
for chi-square test 275
machine-learning 291
stochastic 284
ANOVA 268
formula 268
test 268
Ant colony optimization (ACO) 256, 278, 283, 286, 288, 289, 333
Application 192, 291
data-intensive 192
image-related recognition 291
Approaches 17, 51, 142, 143, 207, 209, 215, 219, 221, 256, 257, 259, 271, 272, 278, 284, 286
backward elimination 259
classic optimization 286
evolutionary computing 284
favored correlation 272
filter-based 17, 256, 257, 259, 271
function optimization 143
value-based reinforcement learning 142
Architecture 11, 13, 104, 140, 174, 183, 221, 228, 241, 246
comprehensive 228
contemporary 221
Artificial intelligence 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 108, 325, 330
branch of 2, 325

evolution of 3, 4, 5
Artificial neural network (ANN) 170, 179, 180, 181, 182, 183, 190, 191, 193, 194, 196, 197, 198, 227, 228
and neural computing 181
Artificial neurons 190, 197, 198, 221
Assumptions for random forest 37
Attributes 24, 325
ordering 24
psychological 325
Automatic handwriting production 179

B

Backpropagation method 192
Batch gradient descent 219
Bayes' theorem 48
Behavior 1, 7, 141, 142, 143, 279, 286
ant colony foraging 286
chromosomal 279
Bellman 148, 150
equations 150
formula 148
Binary 128, 129, 130
classification techniques 128
logistic regression 129, 130
Biological neural network (BNN) 108, 179, 180, 181, 182, 185
Boltzmann machine 174, 176
learning algorithm 176
network 176
Branches, multi-layered neural network 220

C

Calculus 189, 215
differential 215
Camera 302, 308, 309
gazing 309
security 308
Central 180, 265, 266, 318
limit theorem (CLT) 265, 266

nervous system (CNS) 180, 318
 Centroids, cluster's 75
 Classification 21, 22, 23, 24, 32, 49, 67, 93, 127, 129, 184, 309
 and regression trees (CART) 23, 24, 32
 binary 22, 127, 129, 184
 conditional probability model of 49
 lineSVC support vector 67
 multi-class 22, 67
 picture 309
 predict modeling 21
 semantic 93
 Classification algorithms 21, 40, 49, 116
 conditional probability model 49
 non-parametric 40
 Cluster algorithm 89, 100, 101, 102, 103, 104
 Clustering 72, 73, 90, 91, 94, 100, 101, 113, 332
 agglomerative 90, 94
 algorithms 72, 73, 101, 113, 332
 methods, complete link distance 100
 single linkage 91
 Computational 5, 12
 force 12
 intelligence 5
 Computer vision 19, 171, 220, 229, 301
 method 301
 techniques 19
 Computing 2, 150, 152
 machinery and intelligence 2
 Q-values 152
 value function 150
 Conditional probabilities 49, 50, 54, 55, 56, 177
 Connected neural network 174
 Control 4, 180, 192
 motor 180
 sensor 180
 Convolutional neural networks (CNN) 171, 172, 175, 178, 220, 221, 222, 228, 231, 235, 318, 319, 333
 Correlation 13, 43, 189, 256, 270, 271, 272, 273, 276
 analysis 272, 276
 coefficients 272, 273

technique 272
 test 272
 Cuckoo search algorithm 283

D

Dataset, synthetic 87, 100
 Decision rule, binary threshold 177
 Deep learning 2, 16, 170, 171, 172, 174, 179, 255, 290, 291, 301, 326, 331, 332, 333
 algorithms 16, 170, 171, 172, 174, 255, 290, 291, 326, 331, 333
 and machine learning algorithms 301
 application of 179, 291
 techniques 2, 290, 332
 Deep neural network (DNNs) 171, 203, 205, 214, 239
 Defense advanced research projects agency (DARPA) 3
 Density-based methods 73
 Dependent 114, 115, 117, 118, 122, 268, 270, 271
 class variables 115
 variables 114, 115, 117, 118, 122, 268, 270, 271
 Diseases 318
 cerebrovascular 318
 demyelinating 318
 Dynamic treatment regimens (DTRs) 158

E

Experience, sensory 242

F

Features 259, 301
 visual 301
 worst performing 259
 Feature selection 18, 256, 257, 289, 333
 algorithms 18, 256, 289, 333
 methods 256, 257
 techniques 256, 257, 289, 333

Filter 257
and wrapper methods 257
techniques 257
Filter-based feature selection 259, 260, 271
approaches 271
module 259, 260
Financial industry 116
Food 286, 287, 288
source 287, 288
supply 286
Forecast data 115
Forest algorithm 36
Frameworks 4, 5, 7, 8, 10, 12, 147
cognitive 5
expert systems 4
knowledge-based 5
neural architecture model 12
Fraud detection 193

G

Greedy optimization method 259

H

Hopfield network 176, 177, 192, 242, 243, 245

I

Influence training time 12
Information, sensor 12
Input and output layers 171, 186, 199, 241

K

Key 10, 140
motivation 10
terminologies 140
Kohonen neural network 192

L

Learning 6, 133, 216, 331, 333
facet of machine learning and deep 331, 333
programmed dynamic response 6
rate's dynamics 216
technique 133
Learning algorithm 14, 19, 32, 174
dynamic 32
Linear regression 117, 127, 272
analysis 117
correlation coefficient 272
on diabetes dataset 127
technique 117
Linear separation problems 66
Load 126, 134
diabetes dataset 126
logistic regression 134
Logical theory 60
Logistic 114, 115, 116, 127, 128, 129, 130, 131, 132, 133, 134, 137, 205, 332, 333
function 127, 132, 205
regression 114, 115, 116, 127, 128, 129, 130, 131, 132, 133, 134, 137, 332, 333
regression equation 128, 133
Long short term memory (LSTM) 12, 170, 246, 247, 248, 249, 250, 326, 327, 328

M

Machine learning 1, 16, 17, 18, 20, 133, 138, 139, 171, 172, 184, 185, 199, 256, 257, 289, 330, 331, 332, 333
algorithms 16, 18, 20, 133, 138, 139, 171, 172, 256, 257, 331, 332, 333
methods 139, 171, 184, 185, 199
research 139
system 289, 333
techniques 1, 257, 330, 331
technology 17
Magnetic resonance imaging (MRI) 291, 313, 318
Markov's decision process (MDP) 143, 145, 146, 147, 149, 150

Mathematical theory 17
 McCulloch-Pitts neuron 188
 Medical imaging computational linguistic 290
 Memory 7, 237
 restricted 7
 transferring 237
 Methods 73, 259
 data-gathering 259
 grid-based 73
 hierarchical-based 73
 Minkowski distance measure 86
 Model, sentiment analysis 240
 Modeling, predictive regression 116
 Money, superfluous imaging wastes 312
 MRI scanner 318
 Multi 130, 170, 188, 189, 191, 235, 282
 class logistic regression 130
 layer perceptron (MLP) 170, 188, 189, 191,
 235
 objective functions 282
 Multiple feature selection techniques 259

N

Natural language processing (NLP) 18, 171,
 239, 291, 325, 326
 Neural networks 108, 179, 180, 181, 182, 185,
 189, 190, 333
 biological 108, 179, 180, 181, 182, 185
 feed-forward 189, 333
 work 190
 Neuroimaging 290, 291
 Neurons 183, 202
 network's 202
 primary 183
 Null hypothesis 259, 260, 263, 264, 268, 274
 Number 73, 86
 cluster's 86
 finite 73

O

Object recognition technology 309
 Ordinary least squares (OLS) 124

Output 196, 198
 layer neurons 198
 preceding layer's 196

P

Particle swarm optimization (PSO) 256, 278,
 282, 283, 284, 286, 289, 333
 Pattern recognition 242, 290, 291
 system 291
 Pearson's correlation coefficient 17, 256
 Phylogenetic reconstructions 93
 Pictures 5, 9, 10, 176, 200, 203, 220, 222, 223,
 224, 225, 226, 228, 298, 301, 302
 digital 298
 recognition 220
 tiny 176
 visual 9
 Pixels, raw picture 221
 Population, hypothesized 261
 Positron emission tomography (PET) 313
 Print 35, 40, 48, 58, 70, 195, 233, 315, 317,
 318, 329
 accuracy score 35, 40, 48, 58, 70
 characters 317, 318
 data number 233, 329
 faces image 315
 test result 195
 Problems, binary 130
 Process 130, 142, 169, 173, 180, 184, 191,
 215, 225, 227, 257, 258, 280, 282, 292,
 324
 binary classification 130
 decision-making 142
 image data 292
 Processing 18, 171, 175, 179, 197, 227, 237,
 239, 291, 325
 natural language 18, 171, 239, 291, 325
 Programming 5, 149, 150
 banking 5
 dynamic 149, 150
 Programming approach for 33, 38, 69, 100,
 125, 134, 193, 250
 artificial neural network 193

decision tree 33
 hierarchical clustering 100
 linear regression 125
 logistic regression 134
 random forest 38
 recurrent neural network 250
 support vector machine 69
 Python codes 16, 17, 19, 71, 72, 332
 and outputs 72
 snippets 16
 Python implementation 290, 292, 299, 302, 309, 314, 319, 326, 331, 334

R

Radial basis function neural network 191
 Radiological imaging methods 312
 Real-world applications 157, 170, 255, 290, 330, 331, 333, 334
 of machine learning and deep learning algorithms 331, 333
 Receiver operating characteristic (ROC) 22
 Recognition 18, 108, 171, 175, 239, 290, 291, 292, 298, 301, 302, 309, 313, 330, 331, 333
 automatic 291
 facial 291, 331, 333
 learning-based vision 313
 software 292
 speech 108, 171, 239, 330
 statistical system 175
 supported handwriting 298
 Recognition system 22, 298, 302
 facial 22
 vision 302
 Recurrent neural network (RNNs) 170, 171, 172, 174, 175, 176, 234, 235, 236, 237, 239, 242, 254, 255, 333
 Regression 38, 115, 116, 120, 123, 125, 128
 equation 125
 functions 38
 methods 115, 116, 120, 123, 128
 techniques 115, 116
 Regression analysis 114, 115, 116, 137, 332

algorithms 114
 techniques 137, 332
 Reinforcement learning 15, 17, 140, 141, 156, 158, 159, 160
 agent 159
 concept 17
 systems 158
 task 156
 working condition of 141
 Relationships 276, 277
 monotonic 276
 non-monotonic 277
 ReLU 209, 210, 211, 225
 activation function 209, 210, 225
 neurons 211
 RGB color channels 223
 Ridge regression 125, 257
 RL 17, 141, 169, 333
 problem 141
 technology 17, 169, 333
 RNN algorithm 240
 RNN architecture 240, 241
 basic many-to-many 241
 many-to-one 240

S

Schizophrenia 318
 Self-organizing map 17, 72, 104, 105, 106, 107, 108, 113, 172, 332
 neural-network-based 172
 Self organizing neural network 192
 Semi-supervised 14, 15
 learning 14, 15
 machine learning algorithms 15
 Sensory input interpretations 177
 Sentiment analysis 290, 291, 325, 326, 330
 Sigmoid 130, 188, 207, 208, 211
 activation function 188, 207, 208, 211
 function and multi-class logistic regression 130
 Simple RNN unit 236
 Single layer perceptron 186, 191
 Social media analysis 331

Softmax 211, 212
 activation 212
 function's output 211
 mathematical function 212
 Softplus activation function 212, 213
 Soft sign activation function 250
 Solution 36, 60, 72, 150, 154, 229, 270, 278,
 279, 282, 283, 284
 plug-and-play 229
 Spearman rank correlation test 276
 Spearman's correlation measures 276
 Spearman's rank correlation 256, 276, 277,
 289, 333
 coefficient 276
 Speech 87, 325
 data 325
 identification 87
 Standard classification algorithms 21
 Statistical 24, 108, 114, 259, 260, 268
 filter-bases methods 260
 methods 24, 108, 114, 268
 procedure 259
 Stepwise regression 125
 Stochastic gradient descent (SGD) 189, 219,
 220
 Stochasticity 11
 Stock market analysis 116
 Support vector machine(s) (SVM) 12, 17, 19,
 21, 58, 59, 60, 61, 62, 64, 65, 66, 67, 68,
 69, 70, 71, 170, 332
 algorithm 17
 classifier 69
 SVM 58, 59, 64, 66
 algorithm 58, 64, 66
 classifier 59, 66
 kernel 66
 System 14, 15, 20, 26, 29, 72, 104, 108, 157,
 158, 179, 182, 183, 197, 198, 282, 284,
 302, 331
 adaptive knowledge representation 108
 automated video surveillance 331
 biological neural 104
 developing predictive analytics 20
 learning-based object recognition 302
 organic nervous 179

T
 Techniques 66, 73, 107, 124, 125, 127, 178,
 216, 242, 257, 258, 260, 291, 298, 301,
 309, 312, 325
 back-propagation 242
 computational 325
 dimension reduction visualization 107
 hypothesis-testing 260
 multiple gradient descent optimization 216
 neural network-based 178
 Technology 309
 contemporary computer 309
 real-time processing 309
 Telecommunication networks 288
 Test, non-parametric 276
 Text classification functions 51
 Traditional optimization methods 283

U

Ultrasonography 313
 Units 117, 118, 175, 177, 184, 188, 191, 200,
 209, 213, 214, 243, 246
 binary threshold 177
 computational 243
 exponential linear 213, 214
 long short-term memory 246
 neural network's solitary processing 184
 neuronal 175
 non-linear activation 191
 rectified linear 209, 214

V

Validation 294, 320, 321
 directory 294, 320, 321
 folder 294, 321
 Values 75, 81, 116, 128, 187, 190, 215, 258,
 262, 264
 discrete 116
 error 215, 258
 fixed 264

gradient 215
probability 128
random 75, 81, 187
theoretical 262
threshold 190
Vanishing gradient issue 209
Variables 43, 50, 51, 52, 53, 55, 86, 114, 115,
117, 123, 177, 269, 272, 273, 275, 276
composite 269
measurement 268
numerical 55, 272
qualitative 276
quantitative 86
Vectors 61, 66, 124, 130, 131, 175, 176, 192,
211, 212, 215, 227, 245, 270, 273, 279
actual output 245
binary 176
discrete 192
finite-length 279
higher-dimensional 124
one-dimensional 227
single long continuous linear 227
Video 241, 290, 291, 308, 309, 331, 334
categorization 241
classification 309
processing 290, 291, 308, 309, 331, 334
processing for object detection 308
Visual performance 11

X

XOR function 186

W

Working principle of 172, 190
artificial neural network 190
deep learning 172
Working python program 17
World 3, 156
machines 3
real physical 156
Wrapper 256, 257, 258
based approach 256
methods 257, 258



Dr. Indranath Chatterjee

Dr. Indranath Chatterjee is working as a Professor in the Department of Computer Engineering at Tongmyong University, Busan, South Korea. He received his Ph.D. in Computational Neuroscience from the Department of Computer Science, University of Delhi, Delhi, India. His research areas include Computational Neuroscience, Machine Learning, Deep Learning, Schizophrenia, and Medical Imaging. To date, he has authored and edited eight books on Computer Science and Neuroscience published by renowned international publishers. He has published numerous research papers in international journals and conferences. He is a recipient of various global awards in neuroscience. He is currently serving as a Chief Section Editor of a few renowned international journals and serving as a member of the Advisory board and Editorial board of various international journals and Open-Science organizations worldwide. He holds several projects in government & non-government organizations as PI/co-PI, in collaboration with several universities globally. He is an active professional member of the Association of Computing Machinery (ACM, USA), Organization of Human Brain Mapping (OHBM, USA), Federations of European Neuroscience Society (FENS, Belgium), Association for Clinical Neurology and Mental Health (ACNM, India), and International Neuroinformatics Coordinating Facility (INCF, Sweden).