LIVER DISEASE PREDICTION USING MACHINE LEARNING

CO8811 – PROJECT REPORT

Submitted by

BHARATH LAKSHMAN S S 211420118013

in partial fulfillment for the award the degree

of

BACHELOR OF ENGINEERING

in

COMPUTER AND COMMUNICATION ENGINEERING

PANIMALAR ENGINEERING COLLEGE

(An Autonomous Institution, Affiliated to Anna University, Chennai)

MARCH 2024

BONAFIDE CERTIFICATE

Certified that this project report **"LIVER DISEASE PREDICTION USING MACHINE LEARNING"** is the bonafide work of **BHARATH LAKSHMAN S S (211420118013)** who carried out the project work under my supervision.

SIGNATURE

Dr.B.ANNI PRINCY M.E., Ph.D.,

HEAD OF THE DEPARTMENT

PROFESSOR,
COMPUTER AND COMMUNICATION ENGINEERING,

PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI, POONAMALLEE,
CHENNAI- 600123.

SIGNATURE

Dr.R.ANAND BABU.,M.TECH.,Ph.D.,

SUPERVISOR

ASSOCIATE PROFESSOR,
ARTIFICIAL INTELLIGENCE AND MACHINE LEARNING,

PANIMALAR ENGINEERING COLLEGE,
NAZARATHPETTAI, POONAMALLEE,
CHENNAI- 600123.

Certified that the above candidate(s) was/ were examined in the End Semester Project Viva-Voce Examination held on.....

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

We express our deep gratitude to our respected Secretary andCorrespondent **Dr.P.CHINNADURAI, M.A., Ph.D.** for his kind words and enthusiastic motivation, which inspired us a lot in completing this project.

We would like to extend our heartfelt and sincere thanks to our Directors Tmt. **C.VIJAYARAJESWARI, Dr. C. SAKTHIKUMAR, M.E., Ph.D**., and **Dr.SARANYASREE SAKTHIKUMAR B.E., M.B.A., Ph.D**., for providing us with the necessary facilities for completion of this project.

We also express our gratitude to our Principal **Dr.K.MANI, M.E., Ph.D.,** for his timely concern and encouragement provided to us throughout the course.

We thank our HOD of Computer and Communication Engineering Department, **Dr. B. ANNI PRINCY, M.E., Ph.D**., Professor, for the support extended throughout the project.

We would like to thank our supervisor, **Dr.R.ANAND BABU.,M.TECH.,Ph.D.,** Associate Professor, and all the faculty members of the Department of Computer and Communication Engineering for their advice and suggestions for the successful completion of the project.

BHARATH LAKSHMAN S S

ABSTRACT

Liver diseases are becoming one of the most fatal diseases in several countries. Patients with Liver disease have been continuously increasing because of excessive consumption of alcohol, inhale of harmful gases, intake of contaminated food, pickles and drugs. liver patient datasets are investigate for building classification models in order to predict liver disease. This dataset was used to evaluate prediction algorithms in an effort to reduce burden on doctors. In that paper, we proposed as checking the whole patient Liver Disease using Machine learning algorithms. Chronic liver disease refers to disease of the liver which lasts over a period of six months. So in that, we will take results of how much percentage patients get disease as a positive information and negative information. Using classifiers, we are processing Liver Disease percentage and values are showing as a confusion matrix. We proposed a various classification scheme which can effectively improve the classification performance in the situation that training dataset is available. In that dataset, we have nearly 500 patient details. We will get all that details from there. Then we will good and bad values are using machine learning classifier. Feature selection techniques are implemented to identify the most relevant variables contributing to disease prediction. Model performance is evaluated using metrics such as accuracy, sensitivity, specificity, and area under the receiver operating characteristic curve (AUC-ROC).

Additionally, model interpretability is assessed to gain insights into the underlying mechanisms driving the predictions.

The results demonstrate the effectiveness of machine learning models in accurately predicting the risk of liver disease. The proposed approach facilitates early identification of high-risk individuals, enabling timely interventions and personalized healthcare strategies. Furthermore, the interpretability of the models enhances clinical decision-making by providing clinicians with actionable insights into the factors influencing disease onset and progression. Overall, this study underscores the potential of machine learning techniques in improving the prediction and management of liver disease, thereby contributing to better patient outcomes and healthcare resource utilization. Thus outputs shows from proposed classification model indicate that Accuracy in predicting the result.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

PCA	- Principal Component Analysis
t-SNE	- t-distributed stochastic neighbour embedding
SARSA	- State-Action-Reward-State-Action
DDPG	- Deep Deterministic Policy Gradient
GPL	- General Public License
SVM	- Support Vector Machine
GUI	- Graphical User Interface
CDSS	- Clinical Decision Support Systems
ICT	- In-circuit testing

CHAPTER 1

INTRODUCTION

1.1 INTRO

Liver disease is a significant global health issue, affecting millions of individuals worldwide and posing substantial challenges to healthcare systems. Early detection and accurate prediction of liver diseases are essential for timely intervention and improved patient outcomes. With advancements in machine learning techniques and the availability of vast amounts of healthcare data, there is growing interest in leveraging these technologies to develop predictive models for liver disease.

Machine learning offers powerful tools for analyzing complex medical data, identifying patterns, and making predictions based on patient characteristics. By employing machine learning algorithms, healthcare providers can harness the wealth of information contained in electronic health records, medical imaging studies, and laboratory tests to develop accurate and personalized prediction models for liver diseases. By leveraging diverse data sources and advanced predictive analytics techniques, we seek to improve early detection, risk stratification, and treatment planning for patients with liver diseases.

The development of machine learning-based prediction models for liver disease entails several key steps, including data collection, preprocessing, feature selection, model development, evaluation, and deployment. Through a systematic approach, we aim to harness the full potential of machine learning to address the complex challenges associated with liver disease diagnosis and management.

By developing accurate and clinically relevant prediction models, we envision enhancing healthcare providers' ability to identify at-risk individuals, optimize resource allocation, and improve patient outcomes. Ultimately, the integration of machine learning

into liver disease prediction holds promise for transforming healthcare delivery and advancing the field of hepatology.

Anaconda, a popular distribution of the Python programming language, serves several purposes in the context of liver disease prediction using machine learning.

It provides a suite of powerful libraries such as pandas, NumPy, and scikit-learn, which are essential for data analysis and preprocessing. These libraries enable users to load, manipulate, and clean datasets containing liver disease-related data, preparing them for use in machine learning models.

Anaconda serves as a comprehensive platform for liver disease prediction using machine learning, providing the necessary tools, libraries, and resources to analyze data, develop predictive models, and deploy them into production environments. Its versatility, ease of use, and extensive community support make it a valuable asset for researchers, healthcare professionals, and data scientists working in the field of liver disease prediction and beyond.

1.2 Overview

Aim of the project is to create an artificial intelligent model which classifies liver cancer is affected or not using CT/MRI images with deep learning technique.

1.3 Scope of the Project

The main contributions of this project therefore are

- Dataset collection
- Dataset Preprocessing
- Training the Model
- Testing of Dataset

1.4 Existing System

Typically, the existing mechanisms assumed that the accuracy of prediction was achieved. But this wasn't the case then, hence, it must be improved further to increase the classification accuracy. Existing Models based on feature selection and classification raised some issues regarding with training dataset and Test dataset.

Limitation

- Certain approaches being applicable only for small data.
- Certain combination of classifier over fit with data set.
- Some approaches are not adoptable for real time collection of database implementation.

1.5 Proposed System

Machine learning has attracted a huge amount of researches and has been applied in various fields in the world. In medicine, machine learning has proved its power in which it has been employed to solve many emergency problems such as cancer treatment, heart disease, dengue fever diagnosis and so on. In proposed system , we have to import the liver patient dataset (.csv). Then the dataset should be pre-processed and remove the anomalies and full up empty cells in the dataset , so the we can further improve the effective Liver diseases prediction. Then we are Confusion matrix - For getting a better clarity of the no of correct/incorrect predictions by the model ROC-AUC - It considers the rank of the output probabilities and intuitively measures the likelihood that model can distinguish between a positive point and a negative point. (Note: ROC-AUC is typically used for binary classification only). We will use AUC to select the best model among the various machine learning models.

Advantages

- the performance classification of liver based diseases is further improved.
- Time complexity and accuracy can measured by various machine learning models ,so that we can measures different .
- Risky factors can be predicted early by machine learning models.

1.6 Problem Statement

Liver disease is a significant global health concern, with various factors contributing to its prevalence and severity. The problem at hand is to develop a machine learning-based system capable of accurately predicting the likelihood of liver disease in patients based on their medical history, demographic information, and clinical indicators.

1.7 Key Components:

Data Collection:

Gather comprehensive datasets containing relevant patient information, including demographic data, medical history, lifestyle factors, and clinical test results such as liver function tests, imaging studies, and biopsy results.

Data Preprocessing:

Cleanse the collected data to handle missing values, outliers, and inconsistencies. Normalize or scale numerical features and encode categorical variables appropriately to prepare the dataset for model training.

Feature Selection and Engineering: Identify the most informative features that contribute to liver disease prediction. Perform feature engineering techniques to create new features or transform existing ones to enhance the predictive power of the models.

Model Development:

Utilize various machine learning algorithms such as logistic regression, random forests, support vector machines, or deep learning models to build predictive models. Train these models on the preprocessed dataset to learn patterns and relationships between input features and the presence or absence of liver disease.

Model Evaluation:

Evaluate the performance of the trained models using appropriate evaluation metrics such as accuracy, precision, recall, F1-score, and area under the receiver operating characteristic curve (ROC-AUC). Validate the models using techniques like cross-validation to ensure generalizability and robustness.

Monitoring and Maintenance:

Continuously monitor the performance of the deployed model in real-world clinical settings. Incorporate feedback from healthcare professionals and patients to improve the model's accuracy and reliability over time. Update the model periodically with new data and advancements in machine learning techniques.

1.8 Objectives

The primary objective of this project is to develop a machine learning-based system that can assist healthcare providers in identifying individuals at risk of liver disease at an early stage. By leveraging advanced predictive analytics, the system aims to enhance clinical decision-making, optimize resource allocation, and ultimately improve patient outcomes by enabling timely interventions and personalized treatment strategies.

Chapter 2

LITERATURE SURVEY

TITLE 1: Enhanced Preprocessing Approach Using Ensemble Machine Learning Algorithms for Detecting Liver Disease.

Author: Abdul Quadir Md,Sanika Kulkarni,Christy Jackson,Tejas Vaichole

Year: 2021

Abstract

Liver disease is a significant health concern globally, necessitating accurate and efficient diagnostic methods. Traditional preprocessing techniques in liver disease detection often encounter challenges in handling diverse data sources and noise. This paper proposes an enhanced preprocessing approach utilizing ensemble machine learning algorithms to address these limitations.

Limitations

Data Availability and Quality: The efficacy of the proposed approach heavily relies on the availability and quality of datasets. Limited access to high-quality datasets with diverse features may constrain the generalizability of the method.Computational Complexity: Ensemble methods often entail increased computational complexity due to the integration of multiple learners, potentially posing challenges for real-time applications.

TITLE 2: Liver Disease Prediction using Machine learning Classification Techniques.

Author: Ketan Gupta, Nasmin Jiwani, Neda Afreen, Jamia Millia Islamia, Divyarani D.
Year: 2021

Abstract

Liver disease prediction using machine learning classification techniques has garnered significant attention in recent years due to its potential to aid in early diagnosis and intervention. This literature survey provides an overview of existing research in this domain, summarizing key methodologies, findings, and challenges. Various machine learning algorithms, including logistic regression, decision trees, random forests, support vector machines, and neural networks, have been employed to develop predictive models based on clinical and demographic features. Studies have reported promising results, demonstrating the ability of these models to accurately predict liver disease and distinguish between different disease stages. Moreover, feature selection and importance analysis have contributed to identifying relevant predictors and improving model performance.

Limitations

While the literature survey provides valuable insights into liver disease prediction using machine learning classification techniques, several limitations must be acknowledged. Firstly, the scope of the survey may be limited by the inclusion criteria and search strategy employed, potentially overlooking relevant studies published in non-indexed or non-English language journals. Additionally, the interpretation of findings from individual studies may be influenced by factors such as sample size, study population characteristics, and methodological differences in model development.

TITLE 3: The Prediction Of Disease Using Machine Learning.

Author: C K Gomathy, Sri Chandrasekharendra

Year: 2022

Abstract

Application of machine learning for disease prediction. Challenges such as data heterogeneity, imbalanced class distribution, overfitting, and model interpretability hinder the robustness and generalizability of predictive models. Furthermore, the integration of multi-modal data and validation across diverse populations are essential to ensure the reliability and scalability of predictive algorithms. Despite these challenges, machine learning-based disease prediction holds great promise for transforming healthcare delivery by facilitating early diagnosis, risk stratification, and precision medicine approaches. Continued research efforts are warranted to address existing limitations and maximize the clinical impact of predictive analytics in disease management.

Limitations

Most recent advancements or emerging trends in the field, given the rapid pace of innovation in machine learning and healthcare. Furthermore, the generalizability of findings across different diseases, populations, and healthcare settings may be limited, necessitating careful consideration of context-specific factors. Finally, while efforts have been made to summarize key methodologies and limitations, certain nuances or specific challenges associated with disease prediction using machine learning techniques may not be fully addressed. Future research should aim to overcome these limitations and provide more comprehensive insights into the opportunities and challenges in this evolving field of research.

TITLE 4: Mathematical Methods in Medical Imaging: Analysis of Vascular Structures for Liver Surgery Planning.

Author: Selle, D.; Spindler, W.; Preim, B. and Peitgen, H. O

Year: 2000

Abstract

Medical imaging plays a pivotal role in modern healthcare, enabling precise diagnosis and treatment planning. In liver surgery, accurate assessment of vascular structures is crucial for successful outcomes. This literature survey explores mathematical methods utilized in medical imaging for the analysis of vascular structures in the context of liver surgery planning. Various imaging modalities, such as computed tomography (CT), magnetic resonance imaging (MRI), and ultrasound, provide detailed anatomical information. However, extracting and analyzing vascular structures from these images pose significant challenges due to noise, artifacts, and complex geometries. Mathematical techniques, including image segmentation, feature extraction, and computational modeling, are employed to enhance the visualization and quantification of vascular networks.

Limitations

Vascular structures in the liver exhibit intricate geometries and connectivity patterns, which can pose challenges for accurate segmentation and analysis using mathematical methods. Medical imaging modalities may suffer from inherent noise, artifacts, and resolution limitations, affecting the quality of vascular images and subsequently impacting the performance of mathematical algorithms Many existing mathematical techniques require manual intervention for parameter tuning, initialization, or validation, which can be time-consuming and subjective, hindering automation and scalability.

TITLE 5: Coupled Parametric Active Contours". Transactions on pattern Analysis and Machine Intelligence.

Author: Zimmer, C. and Olivo-Marin, J. C

Year: 2005

Abstract

Overview of traditional active contour models (e.g., Snake model) and their limitations in handling noise, weak edges, and complex object boundaries.Discussion on the evolution of parametric active contours and their advantages in terms of computational efficiency and stability.

Review of recent advancements in coupled parametric active contours, including: Techniques for coupling multiple contours to handle multiple objects or complex shapes simultaneously.Integration of additional constraints or prior knowledge to improve segmentation accuracy.Incorporation of machine learning techniques for better feature extraction and classification.

Limitations

Sensitivity to initialization: Many CPAC methods heavily rely on the initial contour placement, leading to suboptimal results if the initial guess is inaccurate.Computational complexity: Despite being more efficient than some traditional methods, CPAC can still be computationally expensive, especially for large-scale images or real-time applications.Limited adaptability to varying image characteristics: CPAC may struggle with images containing highly non-uniform backgrounds, low contrast, or ambiguous object boundaries.Lack of generalization: Some CPAC methods may be tailored to specific applications or image types, limiting their applicability across diverse datasets or scenarios.Challenges in handling occlusions and overlapping objects: CPAC methods may encounter difficulties when objects of interest overlap.

TITLE 6: Provably Powerful Graph Networks

Author: Haggai Maron and Haggai Ben-Hamu

Year: 2022

Abstract

The paper introduces Provably Powerful Graph Networks, demonstrating their capability to represent and process a wide range of graph structures. Leveraging expressive architectures, the model can learn to compute graph functions efficiently. By establishing theoretical guarantees, it ensures robustness and scalability across various domains, including computer vision and molecular chemistry.

Limitations

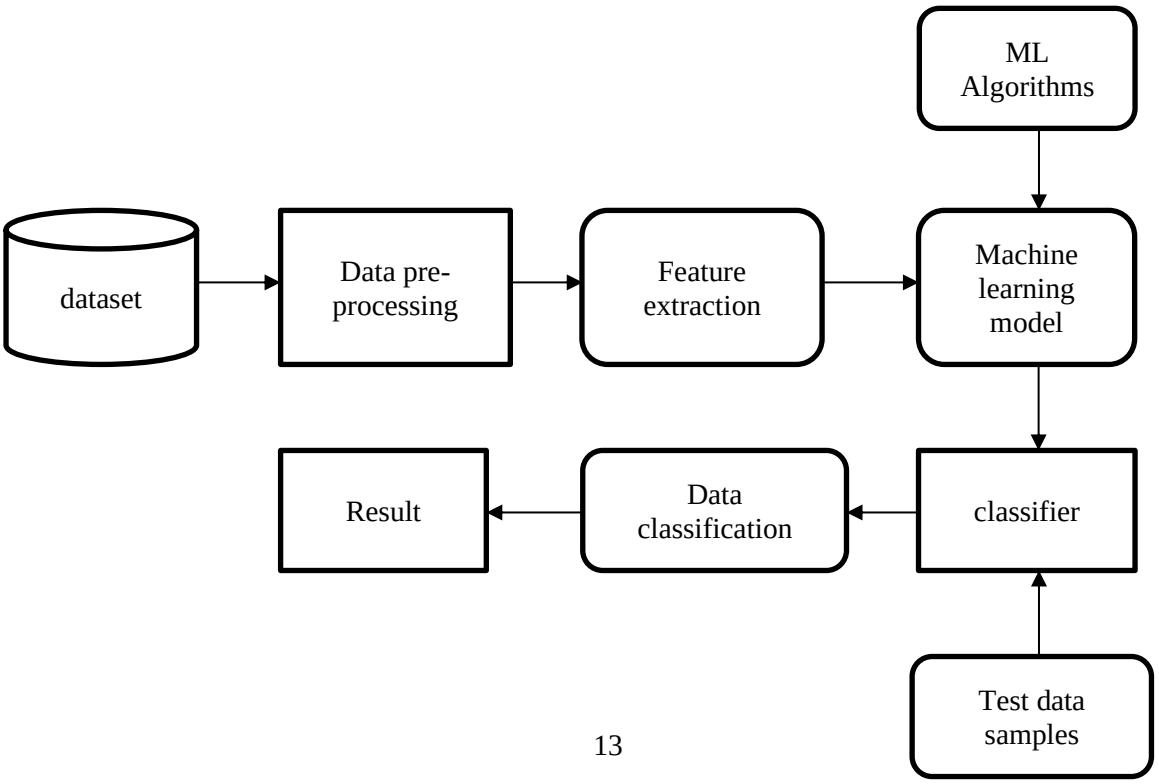
While promising, Provably Powerful Graph Networks still face challenges in scaling to extremely large graphs due to computational complexity. Additionally, their performance might degrade when applied to sparse or noisy data. The model's interpretability could also be a limitation, requiring further research to enhance explainability. Finally, despite theoretical assurances, practical implementation may encounter difficulties in achieving the same level of efficiency and effectiveness across diverse real-world applications.

Chapter 3

SYSTEM DEVELOPMENT

System development typically refers to the process of various steps involved in creating a machine learning-based liver disease prediction system, including data collection, preprocessing, model construction, evaluation, and deployment. The process of developing or enhancing systems—which are collections of related parts cooperating to accomplish a common objective—is commonly referred to as system development. It frequently entails tasks including obtaining stakeholder requirements, creating the system architecture, writing the software, and implementing the system. Systems of many kinds, including hardware, software, and information systems, can be developed using system development techniques.

3.1 Functional Block Diagram of the proposed system



3.2 Modules and module Description

3.2.1 Machine Learning

Machine learning has emerged as a powerful tool in the field of liver disease prediction, offering the potential to improve diagnostic accuracy, identify at-risk individuals, and guide personalized treatment strategies. The methodology typically involves the following steps:

Data Collection

Data used in this paper is a set of records. This step is concerned with selecting the subset of all available data that you will be working with. ML problems start with data preferably, lots of data (examples or observations) for which you already know the target answer. Data for which you already know the target answer is called labelled data.

Data Preprocessing

Organize your selected data by formatting, cleaning and sampling from it.

Three common data pre-processing steps are:

1. Formatting
2. Cleaning
3. Sampling

Formatting: The data you have selected may not be in a format that is suitable for you to work with. The data may be in a relational database and you would like it in a flat file, or the data may be in a proprietary file format and you would like it in a relational database or a text file.

Cleaning: Cleaning data is the removal or fixing of missing data. There may be data instances that are incomplete and do not carry the data you believe you need to address the problem. These instances may need to be removed. Additionally, there may be

sensitive information in some of the attributes and these attributes may need to be anonym zed or removed from the data entirely.

Sampling: There may be far more selected data available than you need to work with. You can take a smaller representative sample of the selected data that may be much faster for exploring and prototyping solutions before considering the whole dataset.

Feature Extraction

Next thing is to do Feature extraction is an attribute extension we created more columns from URL's. Finally, our models are trained using Classifier algorithm. We use the labelled dataset gathered. The rest of our labelled data will be used to evaluate the models. Some machine learning algorithms were used to classify pre-processed data. The chosen classifiers were Random forest.

Evaluation Model

Model Evaluation is an integral part of the model development process. It helps to find the best model that represents our data and how well the chosen model will work in the future.

To avoid over fitting, both methods use a test set (not seen by the model) to evaluate model performance. Performance of each classification model is estimated base on its averaged. **Accuracy** is defined as the percentage of correct predictions for the test data. It can be calculated easily by dividing the number of correct predictions by the number of total predictions.

We predict the accuracy over actual and predicted output and calculate accuracy as –

$$Accuracy = \frac{TP + TN}{TP + TN + FN + FP}$$

3.3 UML DIAGRAMS

Use Case Diagram

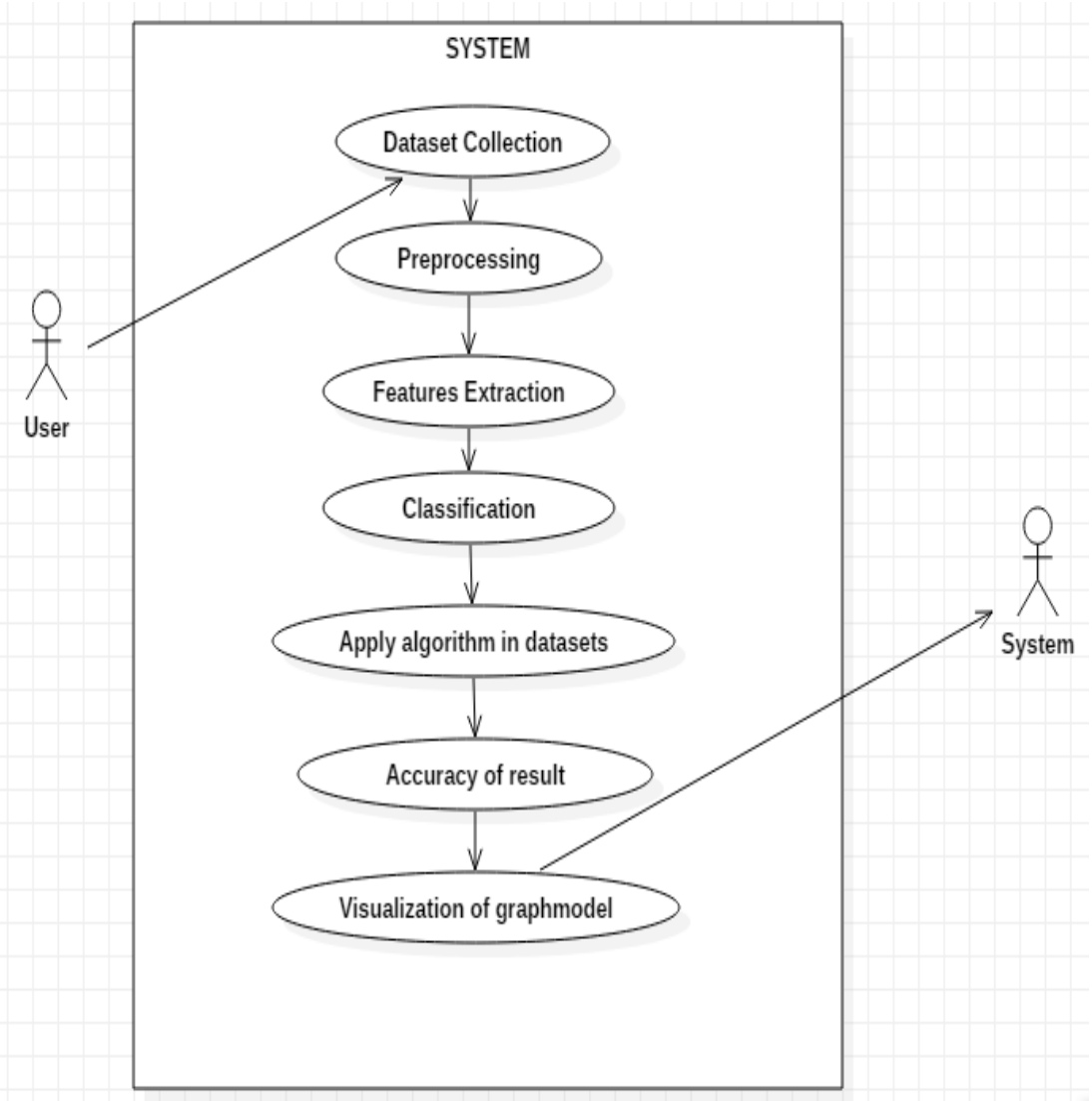


Fig1

Class diagram

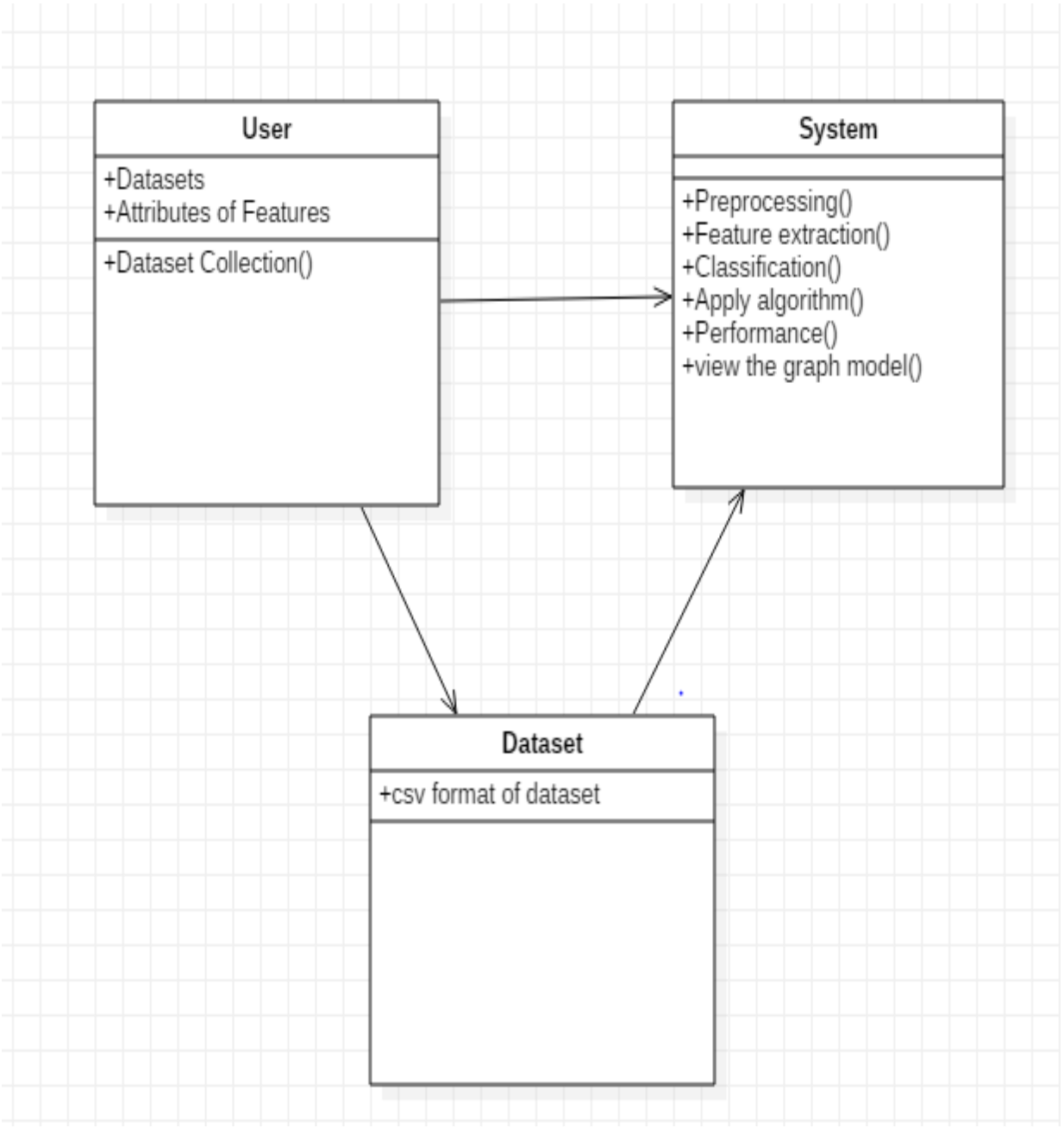


Fig2

Sequence Diagram

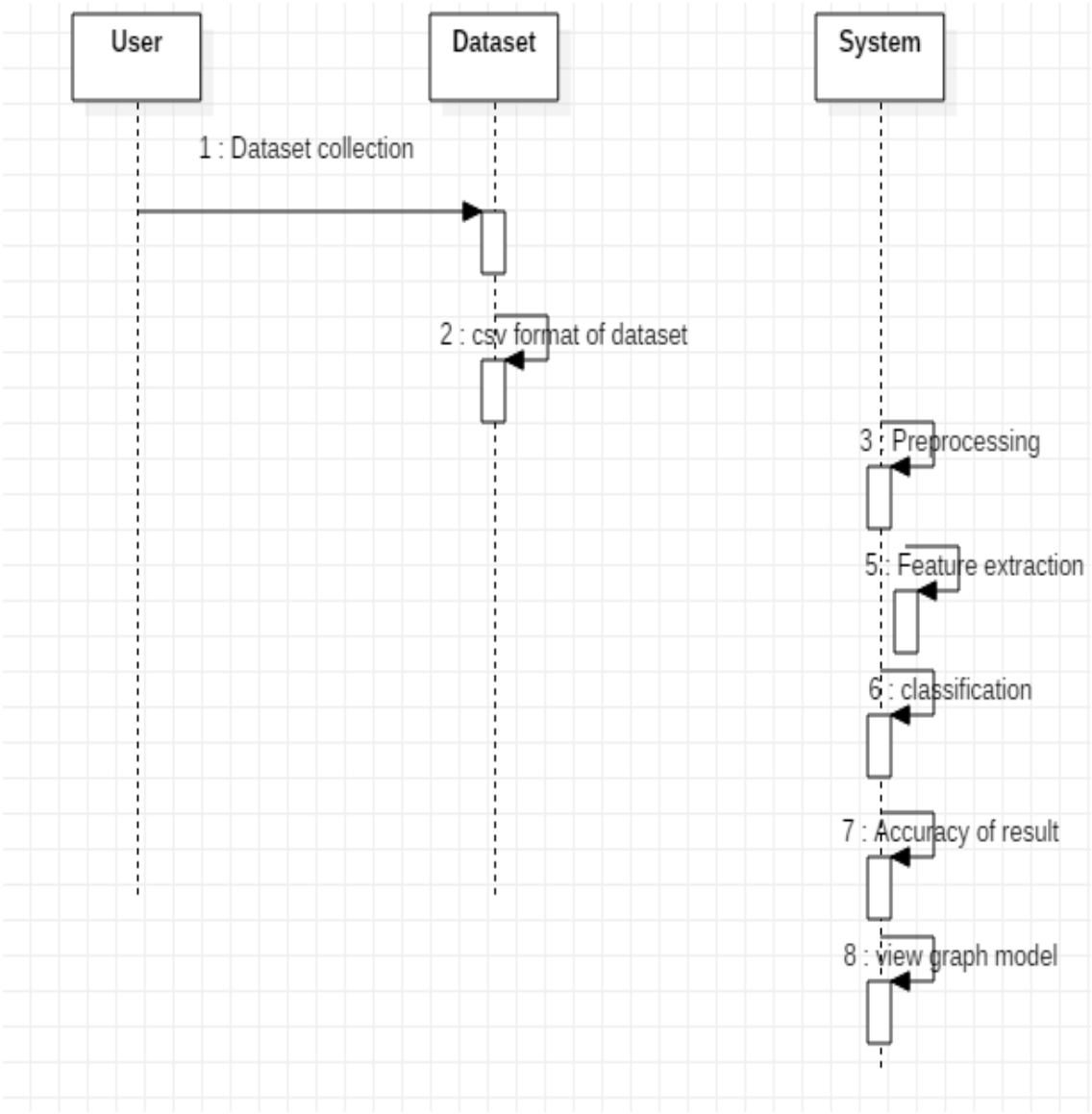


Fig3

Activity Diagram

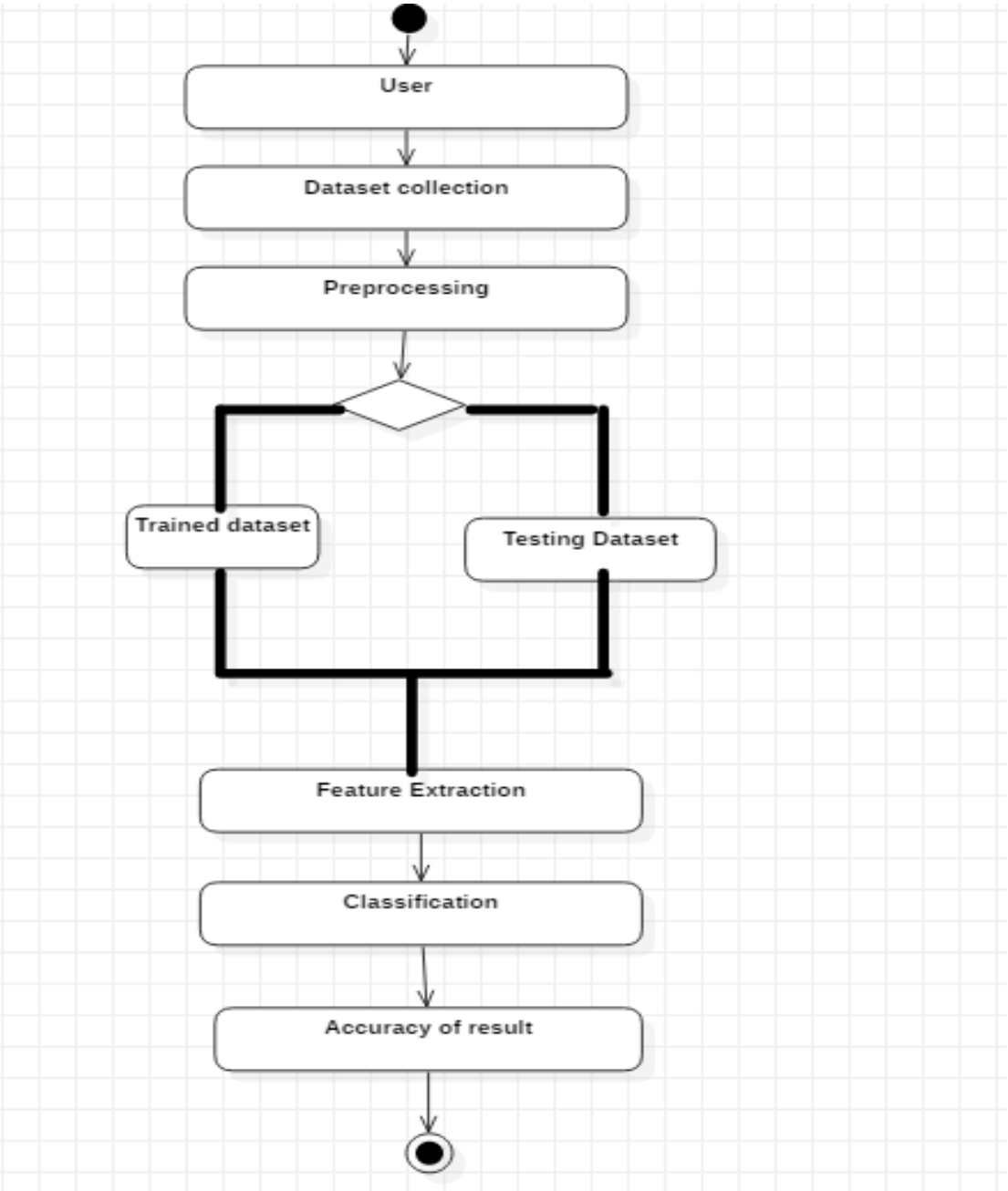


Fig4

Chapter 4

4.1 Performance Analysis

Performance analysis for liver disease prediction using machine learning involves evaluating the predictive models' effectiveness in accurately classifying individuals into different categories

- Evaluation Metrics
- Model Evaluation
- Confusion Matrix
- ROC Curve Analysis
- Precision-Recall Curve
- Interpretation and Comparison
- Validation and Generalization

4.2 Accuracy

The accuracy of liver disease prediction using machine learning, you would typically calculate the accuracy metric based on the model's performance on a test dataset. Here's how you can compute the accuracy:

True Positives (TP): The number of cases where the model correctly predicts the presence of liver disease.

True Negatives (TN): The number of cases where the model correctly predicts the absence of liver disease.

False Positives (FP): The number of cases where the model incorrectly predicts the presence of liver disease when it's actually absent.

False Negatives (FN): The number of cases where the model incorrectly predicts the absence of liver disease when it's actually present.

Using these values, you can compute accuracy using the formula:

Accuracy=TP+TN / TP+TN+FP+FN

Here's an example of how to calculate accuracy using Python:

pythonCopy code

from sklearn.metrics import accuracy_score # Assuming y_true contains the true labels and y_pred contains the predicted labels

```
accuracy = accuracy_score(y_true, y_pred)
print("Accuracy:", accuracy)
```

Replace y_true with the true labels from your test dataset and y_pred with the predicted labels generated by your machine learning model. This code will output the accuracy of the model's predictions.

4.3 Domain Specification

Machine Learning

Machine Learning is a system that can learn from example through self-improvement and without being explicitly coded by programmer. The breakthrough comes with the idea that a machine can singularly learn from the data (i.e., example) to produce accurate results.

Machine learning combines data with statistical tools to predict an output. This output is then used by corporate to makes actionable insights. Machine learning is closely related to data mining and Bayesian predictive modeling. The machine receives data as input,

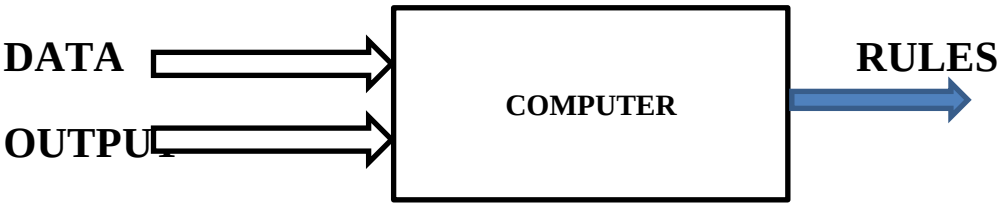
use an algorithm to formulate answers.

A typical machine learning tasks are to provide a recommendation. For those who have a Netflix account, all recommendations of movies or series are based on the user's historical data. Tech companies are using unsupervised learning to improve the user experience with personalizing recommendation.

Machine learning is also used for a variety of task like fraud detection, predictive maintenance, portfolio optimization, automatize task and so on.

Machine Learning vs. Traditional Programming

Traditional programming differs significantly from machine learning. In traditional programming, a programmer code all the rules in consultation with an expert in the industry for which software is being developed. Each rule is based on a logical foundation; the machine will execute an output following the logical statement. When the system grows complex, more rules need to be written. It can quickly become unsustainable to maintain.



Machine Learning

Fig5

How does Machine learning work?

Machine learning is the brain where all the learning takes place. The way the machine learns is similar to the human being. Humans learn from experience. The more we know,

the more easily we can predict. By analogy, when we face an unknown situation, the likelihood of success is lower than the known situation. Machines are trained the same. To make an accurate prediction, the machine sees an example. When we give the machine a similar example, it can figure out the outcome.

The core objective of machine learning is the **learning** and **inference**. First of all, the machine learns through the discovery of patterns. This discovery is made thanks to the **data**. One crucial part of the data scientist is to choose carefully which data to provide to the machine. The list of attributes used to solve a problem is called a **feature vector.** You can think of a feature vector as a subset of data that is used to tackle a problem.

The machine uses some fancy algorithms to simplify the reality and transform this discovery into a **model**. Therefore, the learning stage is used to describe the data and summarize it into a model.

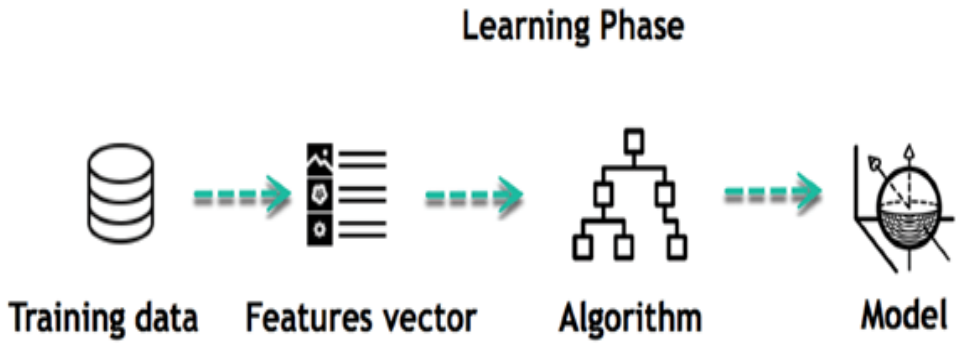


Fig6

For instance, the machine is trying to understand the relationship between the wage of an individual and the likelihood to go to a fancy restaurant. It turns out the machine finds a positive relationship between wage and going to a high-end restaurant: This is the model

Inferring

When the model is built, it is possible to test how powerful it is on never-seen-before data. The new data are transformed into a features vector, go through the model and give a prediction. This is all the beautiful part of machine learning. There is no need to update the rules or train again the model. You can use the model previously trained to make inference on new data.

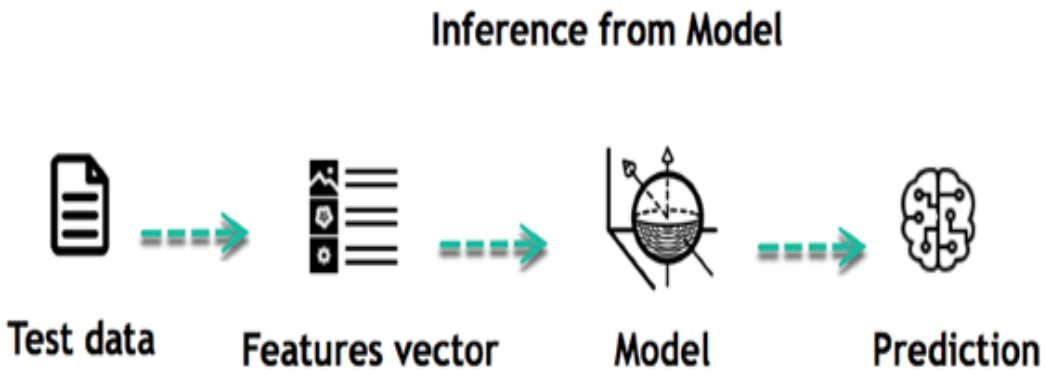


Fig7

The life of Machine Learning programs is straightforward and can be summarized in the following points:

1. Define a question
2. Collect data
3. Visualize data
4. Train algorithm
5. Test the Algorithm
6. Collect feedback
7. Refine the algorithm
8. Loop 4-7 until the results are satisfying

9. Use the model to make a prediction

Once the algorithm gets good at drawing the right conclusions, it applies that knowledge to new sets of data.

Machine learning Algorithms and where they are used?

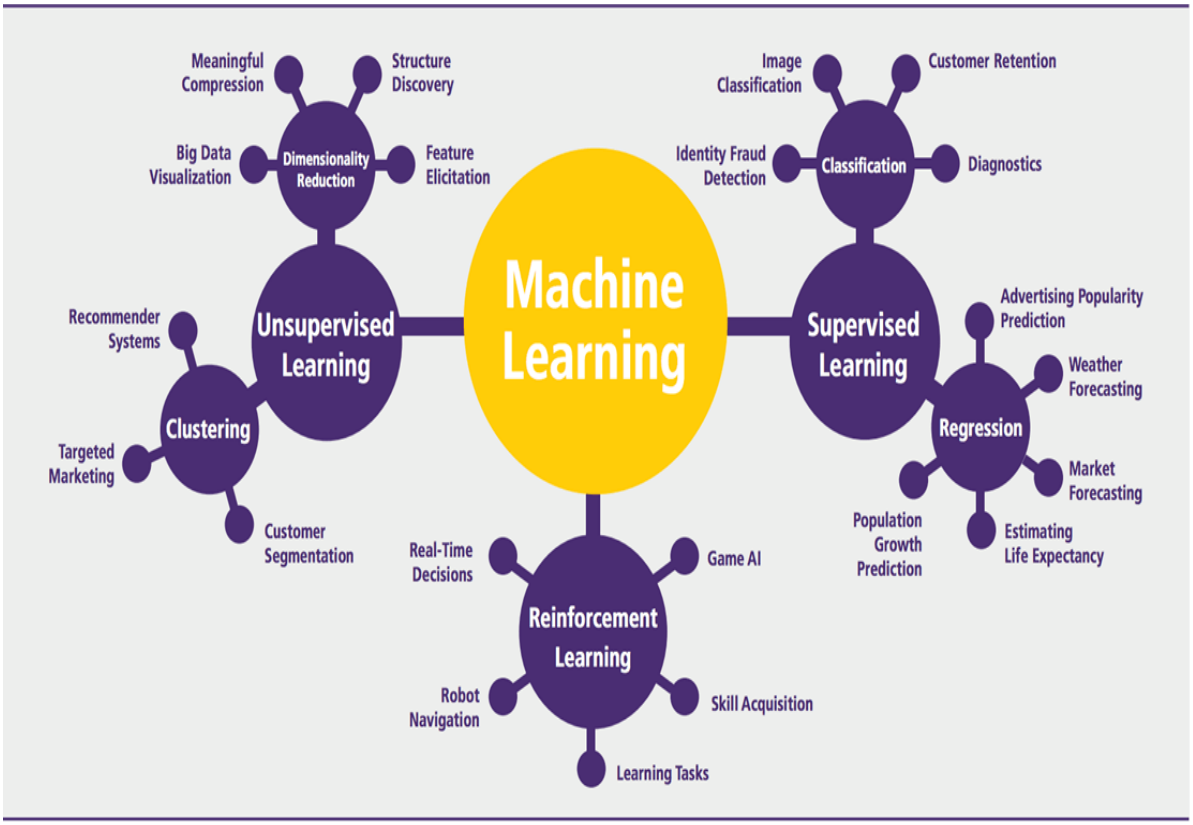


Fig8

Machine learning can be grouped into two broad learning tasks: Supervised and Unsupervised. There are many other algorithms

Supervised learning

An algorithm uses training data and feedback from humans to learn the relationship of given inputs to a given output. For instance, a practitioner can use marketing expense and

weather forecast as input data to predict the sales of cans.You can use supervised learning when the output data is known. The algorithm will predict new data.

There are two categories of supervised learning:

- Classification task
- Regression task

Classification

Imagine you want to predict the gender of a customer for a commercial. You will start gathering data on the height, weight, job, salary, purchasing basket, etc. from your customer database. You know the gender of each of your customer, it can only be male or female. The objective of the classifier will be to assign a probability of being a male or a female (i.e., the label) based on the information (i.e., features you have collected). When the model learned how to recognize male or female, you can use new data to make a prediction. For instance, you just got new information from an unknown customer, and you want to know if it is a male or female. If the classifier predicts male = 70%, it means the algorithm is sure at 70% that this customer is a male, and 30% it is a female.

The label can be of two or more classes. The above example has only two classes, but if a classifier needs to predict object, it has dozens of classes (e.g., glass, table, shoes, etc. each object represents a class)

Regression

When the output is a continuous value, the task is a regression. For instance, a financial analyst may need to forecast the value of a stock based on a range of feature like equity, previous stock performances, macroeconomics index. The system will be trained to estimate the price of the stocks with the lowest possible error.

Algorithm Name	Description	Type
Linear regression	Finds a way to correlate each feature to the output to help predict future values.	Regression
Logistic regression	Extension of linear regression that's used for classification tasks. The output variable 3is binary (e.g., only black or white) rather than continuous (e.g., an infinite list of potential colors)	Classification
Decision tree	Highly interpretable classification or regression model that splits data-feature values into branches at decision nodes (e.g., if a feature is a color, each possible color becomes a new branch) until a final decision output is made	Regression Classification
Naive Bayes	The Bayesian method is a classification method that makes use of the Bayesian theorem. The theorem updates the prior knowledge of an event with the independent probability of each feature that can affect the event.	Regression Classification
Support vector machine	Support Vector Machine, or SVM, is typically used for the classification task. SVM algorithm finds a hyperplane that optimally divided the classes. It is best used with a non-linear solver.	Regression (not very common) Classification
Random	The algorithm is built upon a decision tree to	Regression

forest	improve the accuracy drastically. Random forest generates many times simple decision trees and uses the 'majority vote' method to decide on which label to return. For the classification task, the final prediction will be the one with the most vote; while for the regression task, the average prediction of all the trees is the final prediction.	Classification
AdaBoost	Classification or regression technique that uses a multitude of models to come up with a decision but weighs them based on their accuracy in predicting the outcome	Regression Classification
Gradient-boosting trees	Gradient-boosting trees is a state-of-the-art classification/regression technique. It is focusing on the error committed by the previous trees and tries to correct it.	Regression Classification

Unsupervised learning

In unsupervised learning, an algorithm explores input data without being given an explicit output variable (e.g., explores customer demographic data to identify patterns)

You can use it when you do not know how to classify the data, and you want the algorithm to find patterns and classify the data for you

Algorithm	Description	Type
K-means clustering	Puts data into some groups (k) that each contains data with similar characteristics (as determined by the model, not in advance by humans)	Clustering
Gaussian mixture model	A generalization of k-means clustering that provides more flexibility in the size and shape of groups (clusters	Clustering

Hierarchical clustering	Splits clusters along a hierarchical tree to form a classification system. Can be used for Cluster loyalty-card customer	Clustering
Recommender system	Help to define the relevant data for making a recommendation.	Clustering
PCA/T-SNE	Mostly used to decrease the dimensionality of the data. The algorithms reduce the number of features to 3 or 4 vectors with the highest variances.	Dimension Reduction

Application of Machine learning

Augmentation:

- Machine learning, which assists humans with their day-to-day tasks, personally or commercially without having complete control of the output. Such machine learning is used in different ways such as Virtual Assistant, Data analysis, software solutions. The primary user is to reduce errors due to human bias.

Automation:

- Machine learning, which works entirely autonomously in any field without the need for any human intervention. For example, robots performing the essential process steps in manufacturing plants.

Finance Industry

- Machine learning is growing in popularity in the finance industry. Banks are mainly using ML to find patterns inside the data but also to prevent fraud.

Government organization

- The government makes use of ML to manage public safety and utilities. Take the example of China with the massive face recognition. The government uses Artificial intelligence to prevent jaywalker.

Healthcare industry

- Healthcare was one of the first industry to use machine learning with image detection.

Marketing

- Broad use of AI is done in marketing thanks to abundant access to data. Before the age of mass data, researchers develop advanced mathematical tools like Bayesian analysis to estimate the value of a customer. With the boom of data, marketing department relies on AI to optimize the customer relationship and marketing campaign.

Deep Learning

Deep learning is a computer software that mimics the network of neurons in a brain. It is a subset of machine learning and is called deep learning because it makes use of deep neural networks. The machine uses different layers to learn from the data. The depth of the model is represented by the number of layers in the model. Deep learning is the new state of the art in term of AI. In deep learning, the learning phase is done through a

neural network.

Reinforcement Learning

Reinforcement learning is a subfield of machine learning in which systems are trained by receiving virtual "rewards" or "punishments," essentially learning by trial and error. Google's DeepMind has used reinforcement learning to beat a human champion in the Go games. Reinforcement learning is also used in video games to improve the gaming experience by providing smarter bot.

One of the most famous algorithms are:

- Q-learning
- Deep Q network
- State-Action-Reward-State-Action (SARSA)
- Deep Deterministic Policy Gradient (DDPG)

Difference between Machine Learning and Deep Learning

	Machine Learning	**Deep Learning**
Data Dependencies	Excellent performances on a small/medium dataset	Excellent performance on a big dataset

Hardware dependencies	Work on a low-end machine.	Requires powerful machine, preferably with GPU: DL performs a significant amount of matrix multiplication
Feature engineering	Need to understand the features that represent the data	No need to understand the best feature that represents the data
Execution time	From few minutes to hours	Up to weeks. Neural Network needs to compute a significant number of weights
Interpretability	Some algorithms are easy to interpret (logistic, decision tree), some are almost impossible (SVM, XGBoost)	Difficult to impossible

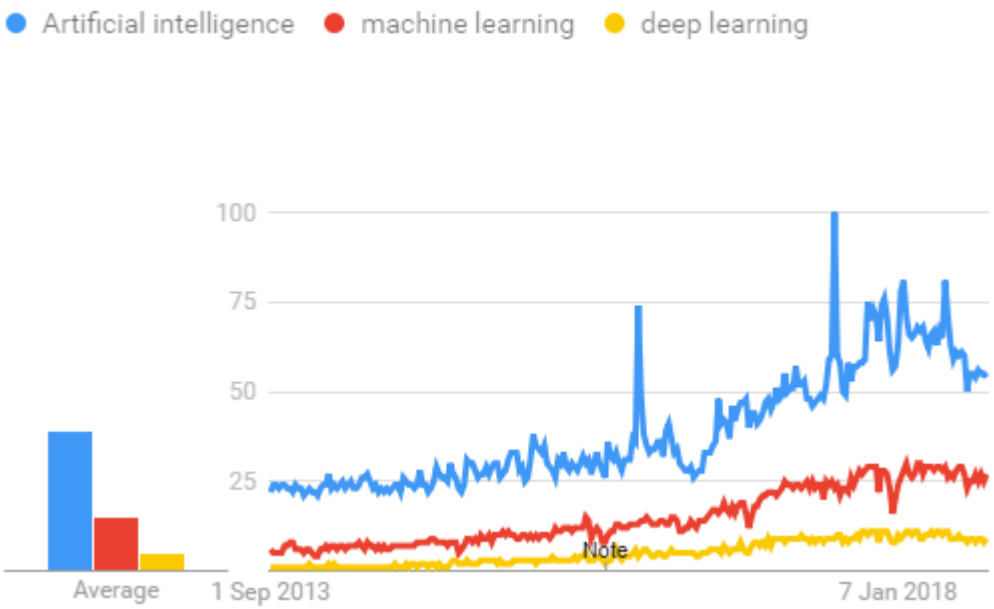
When to use ML or DL?

In the table below, we summarize the difference between machine learning and deep learning.

	Machine learning	**Deep learning**

Training dataset	Small	Large
Choose features	Yes	No
Number of algorithms	Many	Few
Training time	Short	Long

With machine learning, you need fewer data to train the algorithm than deep learning. Deep learning requires an extensive and diverse set of data to identify the underlying structure. Besides, machine learning provides a faster-trained model. Most advanced deep learning architecture can take days to a week to train. The advantage of deep learning over machine learning is it is highly accurate. You do not need to understand what features are the best representation of the data; the neural network learned how to select critical features. In machine learning, you need to choose for yourself what features to include in the model.



4.4 TensorFlow

The most famous deep learning library in the world is Google's TensorFlow. Google product uses machine learning in all of its products to improve the search engine, translation, image captioning or recommendations.

To give a concrete example, Google users can experience a faster and more refined the search with AI. If the user types a keyword a the search bar, Google provides a recommendation about what could be the next word.

Google wants to use machine learning to take advantage of their massive datasets to give users the best experience. Three different groups use machine learning:

- Researchers
- Data scientists
- Programmers.

They can all use the same toolset to collaborate with each other and improve their efficiency.

Google does not just have any data; they have the world's most massive computer, so TensorFlow was built to scale. TensorFlow is a library developed by the Google Brain Team to accelerate machine learning and deep neural network research.

It was built to run on multiple CPUs or GPUs and even mobile operating systems, and it has several wrappers in several languages like Python, C++ or Java.

In this tutorial, you will learn

TensorFlow Architecture

Tensorflow architecture works in three parts:

- Preprocessing the data
- Build the model
- Train and estimate the model

It is called Tensorflow because it takes input as a multi-dimensional array, also known as **tensors**. You can construct a sort of **flowchart** of operations (called a Graph) that you want to perform on that input. The input goes in at one end, and then it flows through this system of multiple operations and comes out the other end as output.

This is why it is called TensorFlow because the tensor goes in it flows through a list of operations, and then it comes out the other side.

Where can Tensorflow run?

TensorFlow can hardware, and software requirements can be classified into

Development Phase: This is when you train the mode. Training is usually done on your Desktop or laptop.

Run Phase or Inference Phase: Once training is done Tensorflow can be run on many different platforms. You can run it on

- Desktop running Windows, macOS or Linux
- Cloud as a web service
- Mobile devices like iOS and Android

You can train it on multiple machines then you can run it on a different machine, once you have the trained model.

The model can be trained and used on GPUs as well as CPUs. GPUs were initially designed for video games. In late 2010, Stanford researchers found that GPU was also very good at matrix operations and algebra so that it makes them very fast for doing these kinds of calculations. Deep learning relies on a lot of matrix multiplication. TensorFlow is very fast at computing the matrix multiplication because it is written in C++. Although it is implemented in C++, TensorFlow can be accessed and controlled by other languages mainly, Python.

List of Prominent Algorithms supported by TensorFlow

- Linear regression: tf.estimator.LinearRegressor
- Classification:tf.estimator.LinearClassifier
- Deep learning classification: tf.estimator.DNNClassifier
- Deep learning wipe and deep: tf.estimator.DNNLinearCombinedClassifier
- Booster tree regression: tf.estimator.BoostedTreesRegressor
- Boosted tree classification: tf.estimator.BoostedTreesClassifier

4.5 Python Overview

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted:** Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

- **Python is Interactive:** You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

- **Python is Object-Oriented:** Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

- **Python is a Beginner's Language:** Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-

68, SmallTalk, Unix shell, and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include:

- **Easy-to-learn:** Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read:** Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain:** Python's source code is fairly easy-to-maintain.
- **A broad standard library:** Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode:** Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable:** Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable:** You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases:** Python provides interfaces to all major commercial databases.

- **GUI Programming:** Python supports GUI applications that can be created and ported to many system calls, libraries, and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

- **Scalable:** Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below:

- IT supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- IT supports automatic garbage collection.

It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python Environment

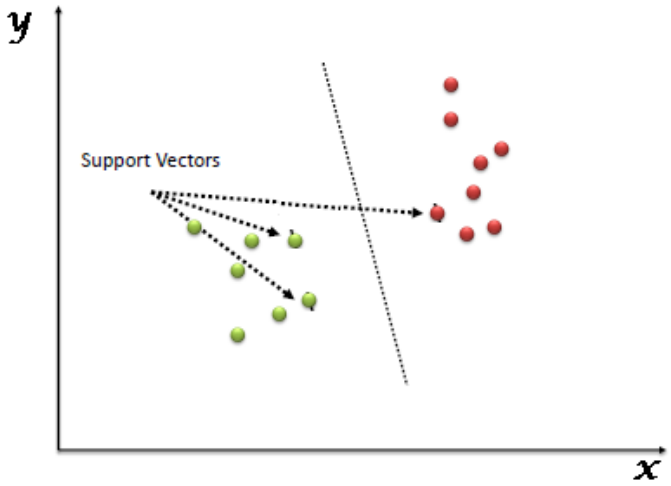
Python is available on a wide variety of platforms including Linux and Mac OS X. Let's understand how to set up our Python environment.

4.6 ALGORITHM

Support Vector Machine:

“Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in

classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).



Support Vectors are simply the co-ordinates of individual observation. The SVM classifier is a frontier which best segregates the two classes (hyper-plane/ line).

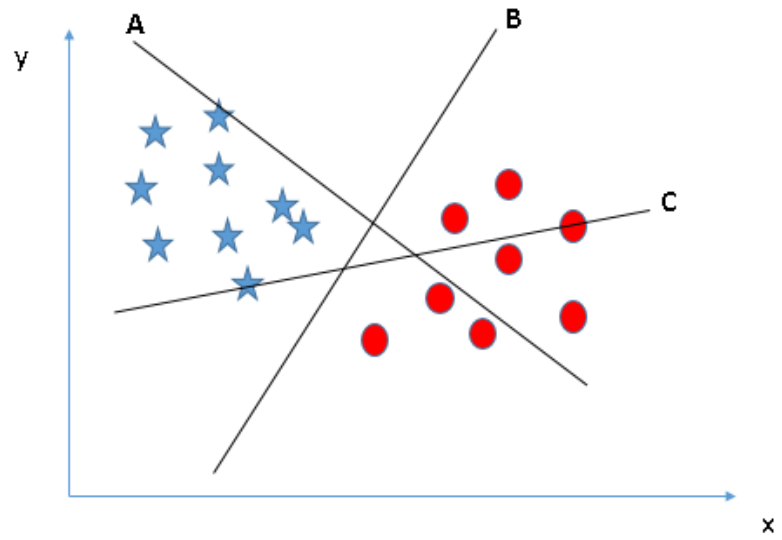
You can look at support vector machines and a few examples of its working here.

How does it work?

Above, we got accustomed to the process of segregating the two classes with a hyper-plane. Now the burning question is “How can we identify the right hyper-plane?”. Don’t worry, it’s not as hard as you think!

Let’s understand:

- **Identify the right hyper-plane :** Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.



- You need to remember a thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.

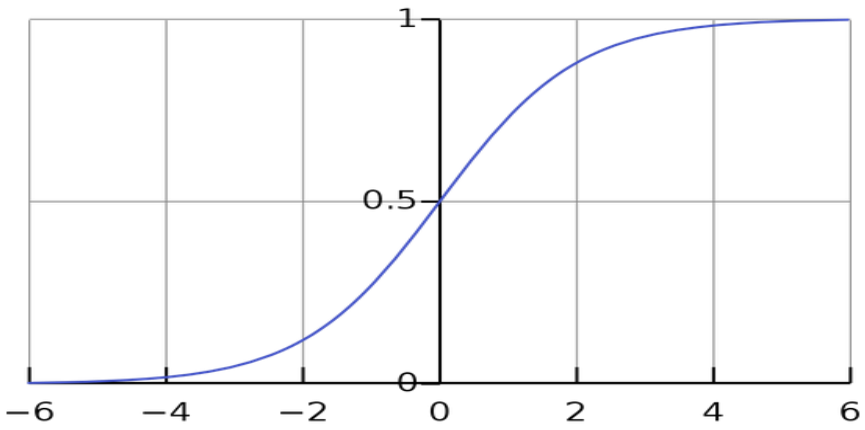
Logistic Regression

Logistic regression is a statistical technique used to predict probability of binary response based on one or more independent variables. It means that, given a certain factors, logistic regression is used to predict an outcome which has two values such as 0 or 1, pass or fail, yes or no etc.

Working of Logistic regression.

Probabilities are estimated using logistic/sigmoid function. The graph of sigmoid

function is an ‘S’ curve.



The mathematical expression is given by

$F(z)=1/1-e-z$

where $z=w0+w1\cdot x1+w2\cdot x2+\ldots+wn\cdot xn$.

Here $w0,w1,w2,...,wn$ are the regression co-efficients of the model and are calculated by Maximum Likelihood Estimation and $x1,x2,x3,...,xn$ are the features or independent variables. $F(z)$ calculates the probability of the binary outcome and using the probabilities we classify the given data point(x) into one of the two.

Artificial Neural Networks

Artificial Neural Network is one of the most used technique for prediction models, ANN is based on the structure and features of Neural Networks, the imitation of human brain. In this the primary computational units are called as neurons, these neurons are connected together in layers, where the data is passed in as the input the network is trained throughout with special equations called as the activation functions. The applications of neural network is widely used for agricultural practicesEven if the statistics are complex, multivariate, nonlinear this community offers the correct

effects and also without any of underlying concepts the relationship between them and the output is extracted.

4.7 RANDOM FOREST

Random forest is a type of supervised machine learning algorithm based on ensemble learning. Ensemble learning is a type of learning where you join different types of algorithms or same algorithm multiple times to form a more powerful prediction model. The random forest algorithm combines multiple algorithm of the same type i.e. multiple decision *trees*, resulting in a *forest of trees*, hence the name "Random Forest". The random forest algorithm can be used for both regression and classification tasks.

HOW RANDOM FOREST WORKS

The following are the basic steps involved in performing the random forest algorithm

1. Pick N random records from the dataset.
2. Build a decision tree based on these N records.
3. Choose the number of trees you want in your algorithm and repeat steps 1 and 2.
4. For classification problem, each tree in the forest predicts the category to which the new record belongs. Finally, the new record is assigned to the category that wins the majority vote.

ADVANTAGES OF USING RANDOM FOREST

pros of using random forest for classification and regression.

1. The random forest algorithm is not biased, since, there are multiple trees and each tree is trained on a subset of data. Basically, the random forest algorithm relies on the power of "the crowd"; therefore, the overall biasedness of the algorithm is reduced.
2. This algorithm is very stable. Even if a new data point is introduced in the dataset the overall algorithm is not affected much since new data may impact one tree, but it is very hard for it to impact all the trees.
3. The random forest algorithm works well when you have both categorical and numerical features.
4. The random forest algorithm also works well when data has missing values or it has not been scaled we.

Chapter 5

SYSTEM REQUIREMENTS AND COMPONENTS

5.1 SYSTEM REQUIREMENTS

Hardware:

1. Windows 7,8,10 64 bit
2. RAM 4GB

Software:

1. Data Set
2. Python 2.7
3. Anaconda Navigator

Python’s standard library

- Pandas
- Numpy
- Sklearn
- seaborn
- matplotlib

5.2 ANACONDA NAVIGATOR

Anaconda Navigator is a desktop graphical user interface (GUI) included in Anaconda distribution that allows you to launch applications and easily manage conda

packages, environments and channels without using command-line commands. Navigator can search for packages on Anaconda Cloud or in a local Anaconda Repository. It is available for Windows, mac OS and Linux.

Why use Navigator?

In order to run, many scientific packages depend on specific versions of other packages. Data scientists often use multiple versions of many packages, and use multiple environments to separate these different versions.

The command line program conda is both a package manager and an environment manager, to help data scientists ensure that each version of each package has all the dependencies it requires and works correctly.

Navigator is an easy, point-and-click way to work with packages and environments without needing to type conda commands in a terminal window. You can use it to find the packages you want, install them in an environment, run the packages and update them, all inside Navigator.

WHAT APPLICATIONS CAN I ACCESS USING NAVIGATOR?

The following applications are available by default in Navigator:

- JupyterLab
- Jupyter Notebook
- QTConsole
- Spyder
- VSCode
- Glueviz
- Orange 3 App

- Rodeo
- RStudio

Advanced conda users can also build your own Navigator applications

How can I run code with Navigator?

The simplest way is with Spyder. From the Navigator Home tab, click Spyder, and write and execute your code.

You can also use Jupyter Notebooks the same way. Jupyter Notebooks are an increasingly popular system that combine your code, descriptive text, output, images and interactive interfaces into a single notebook file that is edited, viewed and used in a web browser.

What’s new in 1.9?

- Add support for **Offline Mode** for all environment related actions.
- Add support for custom configuration of main windows links.
- Numerous bug fixes and performance enhancements.

5.3 PYTHON

Python

Python is a general-purpose, versatile and popular programming language. It's great as a first language because it is concise and easy to read, and it is also a good language to have in any programmer's stack as it can be used for everything from web development to soitware development and scientific applications.

It has simple easy-to-use syntax, making it the perfect language for someone trying to learn computer programming for the first time.

Features of Python

A simple language which is easier to learn, Python has a very simple and elegant syntax. It's much easier to read and write Python programs compared to other languages like: C++, Java, C#. Python makes programming fun and allows you to focus on the solution rather than syntax. If you are a newbie, it's a great choice to start your journey with Python.

- **Free and open source**

  You can freely use and distribute Python, even for commercial use. Not only can you use and distribute software's written in it, you can even make changes to the Python's source code. Python has a large community constantly improving it in each iteration.

- **Portability**
- You can move Python programs from one platform to another, and run it without any changes.

  It runs seamlessly on almost all platforms including Windows, Mac OS X and Linux.

- **ExtensibleandEmbeddable**

  Suppose an application requires high performance. You can easily combine pieces of C/C++ or other languages with Python code. This will give your application high performance as well as scripting capabilities which other languages may not provide out of the box.

- **A high-level, interpreted language**

  Unlike C/C++, you don't have to worry about daunting tasks like memory management,garbagecollectionandsoon Likewise, when you run Python code, it automatically converts your code to the language your computer understands. You

don't need to worry about any lower level operations.

- **Large standard libraries to solve common tasks**
  Python has a number of standard libraries which makes life of a programmer much easier since you don't have to write all the code yourself. For example: Need to connect MySQL database on a Web server You can use MySQLdb library using import MySQL db Standard libraries in Python are well tested and used by hundreds of people. So you can be sure that it won't break your application.
- **Object-oriented**

  Everything in Python is an object. Object oriented programming (OOP) helps you solve a complex problem intuitively. With OOP, you are able to divide these complex problems into smaller sets by creating object

5.4 NUMPY

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. At the core of the NumPy package, is the ndarray object. This encapsulates n-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance. There are several important differences between NumPy arrays and the standard Python sequences: • NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original. • The elements in a NumPy array are all required to be of the same data type, and thus will be the same size

in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements. • NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.

In some ways, NumPy is simply the application of this experience to the Python language – thus many of the operations described in NumPy work the way they do because experience has shown that way to be a good one, in a variety of contexts. The languages which were used to guide the development of NumPy include the infamous APL family of languages, Basis, MATLAB, FORTRAN, S and S+, and others. This heritage will be obvious to users of NumPy who already have experience with these other languages. This tutorial, however, does not assume any such background, and all that is expected of the reader is a reasonable working knowledge of the standard Python language. This document is the "official" documentation for NumPy. It is both a tutorial and the most authoritative source of information about NumPy with the exception of the source code. The result of various manipulations can be displayed simply since the data set has a natural graphical representation. All users of NumPy, whether interested in image processing or not, are encouraged to follow the tutorial with a working NumPy installation at their side, testing the examples, and, more importantly, transferring the understanding gained by working on images to their specific domain. The best way to learn is by doing – the aim of this tutorial is to guide you along this "doing."

5.5 Testing

The various levels of testing are

1. White Box Testing
2. Black Box Testing
3. Unit Testing
4. Functional Testing

5. Performance Testing
6. Integration Testing
7. Objective
8. Integration Testing
9. Validation Testing
10. System Testing
11. Structure Testing
12. Output Testing
13. User Acceptance Testing

White Box Testing

White-box testing (also known as **clear box testing**, **glass box testing**, **transparent box testing**, and **structural testing**) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT).

White-box test design techniques include:

- Control flow testing
- Data flow testing
- Branch testing
- Path testing
- Statement coverage
- Decision coverage

The whole point of white-box testing is the ability to know which line of the code is being executed and being able to identify what the correct output should be.

Levels

1. Unit testing. White-box testing is done during unit testing to ensure that the code is working as intended, before any integration happens with previously tested code. White-box testing during unit testing catches any defects early on and aids in any defects that happen later on after the code is integrated with the rest of the application and therefore prevents any type of errors later on.
2. Integration testing. White-box testing at this level are written to test the interactions of each interface with each other. The Unit level testing made sure that each code was tested and working accordingly in an isolated environment and integration examines the correctness of the behaviour in an open environment through the use of white-box testing for any interactions of interfaces that are known to the programmer.
3. Regression testing. White-box testing during regression testing is the use of recycled white-box test cases at the unit and integration testing levels.

Black Box Testing

Black-box testing is a method of software testing that examines the functionality of an application (e.g. what the software does) without peering into its internal structures or workings (see white-box testing). This method of test can be applied to virtually every level of software testing: unit, integration,system and acceptance. It typically comprises most if not all higher level testing, but can also dominate unit testing as well

Test procedures

Specific knowledge of the application's code/internal structure and programming

knowledge in general is not required. The tester is aware of *what* the software is supposed to do but is not aware of *how* it does it. For instance, the tester is aware that a particular input returns a certain, invariable output but is not aware of *how* the software produces the output in the first place.

Test cases

Test cases are built around specifications and requirements, i.e., what the application is supposed to do. Test cases are generally derived from external descriptions of the software, including specifications, requirements and design parameters. Although the tests used are primarily *functional* in nature, *non-functional* tests may also be used. The test designer selects both valid and invalid inputs and determines the correct output without any knowledge of the test object's internal structure.

Test design techniques

Typical black-box test design techniques include:

- Decision table testing
- All-pairs testing
- State transition tables
- Equivalence partitioning
- Boundary value analysis

Unit testing

In computer programming, **unit testing** is a method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures are tested to determine if they are fit for use.

It is also essential to implement a sustainable process for ensuring that test case failures are reviewed daily and addressed immediately if such a process is not

implemented and ingrained into the team's workflow, the application will evolve out of sync with the unit test suite, increasing false positives and reducing the effectiveness of the test suite.

Unit testing embedded system software presents a unique challenge: Since the software is being developed on a different platform than the one it will eventually run on, you cannot readily run a test program in the actual deployment environment, as is possible with desktop programs.[7]

Functional testing

Functional testing is a quality assurance (QA) process and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing). Functional Testing usually describes *what* the system does.

Functional testing typically involves five steps .The identification of functions that the software is expected to perform

1. The creation of input data based on the function's specifications
2. The determination of output based on the function's specifications
3. The execution of the test case
4. The comparison of actual and expected outputs

Performance testing

In software engineering, performance testing is in general testing performed to determine how a system performs in terms of responsiveness and stability under a particular workload. It can also serve to investigate, measure, validate or verify other quality attributes of the system, such as scalability, reliability and resource usage.

Performance testing is a subset of performance engineering, an emerging computer science practice which strives to build performance into the implementation, design and architecture of a system.

Testing types

Load testing

Load testing is the simplest form of performance testing. A load test is usually conducted to understand the behaviour of the system under a specific expected load. This load can be the expected concurrent number of users on the application performing a specific number of transactions within the set duration. This test will give out the response times of all the important business critical transactions. If the database, application server, etc. are also monitored, then this simple test can itself point towards bottlenecks in the application software.

Stress testing

Stress testing is normally used to understand the upper limits of capacity within the system. This kind of test is done to determine the system's robustness in terms of extreme load and helps application administrators to determine if the system will perform sufficiently if the current load goes well above the expected maximum.

Soak testing

Soak testing, also known as endurance testing, is usually done to determine if the system can sustain the continuous expected load. During soak tests, memory utilization is monitored to detect potential leaks. Also important, but often overlooked is performance degradation. That is, to ensure that the throughput and/or response times after some long period of sustained activity are as good as or better than at the beginning of the test. It essentially involves applying a significant load to a system for an extended,

significant period of time. The goal is to discover how the system behaves under sustained use.

Spike testing

Spike testing is done by suddenly increasing the number of or load generated by, users by a very large amount and observing the behaviour of the system. The goal is to determine whether performance will suffer, the system will fail, or it will be able to handle dramatic changes in load.

Configuration testing

Rather than testing for performance from the perspective of load, tests are created to determine the effects of configuration changes to the system's components on the system's performance and behaviour. A common example would be experimenting with different methods of load-balancing.

Isolation testing

Isolation testing is not unique to performance testing but involves repeating a test execution that resulted in a system problem. Often used to isolate and confirm the fault domain.

Integration testing

Integration testing (sometimes called **integration and testing**, abbreviated **I&T**) is the phase in software testing in which individual software modules are combined and tested as a group. It occurs after unit testing and before validation testing. Integration testing takes as its input modules that have been unit tested, groups them in larger aggregates, applies tests defined in an integration test plan to those aggregates, and delivers as its output the integrated system ready for system testing.

Purpose

The purpose of integration testing is to verify functional, performance, and reliability requirements placed on major design items. These "design items", i.e. assemblages (or groups of units), are exercised through their interfaces using black box testing, success and error cases being simulated via appropriate parameter and data inputs. Simulated usage of shared data areas and inter-process communication is tested and individual subsystems are exercised through their input interface.

Test cases are constructed to test whether all the components within assemblages interact correctly, for example across procedure calls or process activations, and this is done after testing individual modules, i.e. unit testing. The overall idea is a "building block" approach, in which verified assemblages are added to a verified base which is then used to support the integration testing of further assemblages.

Some different types of integration testing are big bang, top-down, and bottom-up. Other Integration Patterns are: Collaboration Integration, Backbone Integration, Layer Integration, Client/Server Integration, Distributed Services Integration and High-frequency Integration.

Big Bang

In this approach, all or most of the developed modules are coupled together to form a complete software system or major part of the system and then used for integration testing. The Big Bang method is very effective for saving time in the integration testing process. However, if the test cases and their results are not recorded properly, the entire integration process will be more complicated and may prevent the testing team from achieving the goal of integration testing.

A type of Big Bang Integration testing is called **Usage Model testing**. Usage Model Testing can be used in both software and hardware integration testing. The basis behind this type of integration testing is to run user-like workloads in integrated user-like environments

Usage Model testing takes an optimistic approach to testing, because it expects to have few problems with the individual components. The strategy relies heavily on the component developers to do the isolated unit testing for their product. The goal of the strategy is to avoid redoing the testing done by the developers, and instead flesh-out problems caused by the interaction of the components in the environment.

Top-down and Bottom-up

Bottom Up Testing is an approach to integrated testing where the lowest level components are tested first, then used to facilitate the testing of higher level components. The process is repeated until the component at the top of the hierarchy is tested.

All the bottom or low-level modules, procedures or functions are integrated and then tested. After the integration testing of lower level integrated modules, the next level of modules will be formed and can be used for integration testing. This approach is helpful only when all or most of the modules of the same development level are ready. This method also helps to determine the levels of software developed and makes it easier to report testing progress in the form of a percentage.

Top Down Testing is an approach to integrated testing where the top integrated modules are tested and the branch of the module is tested step by step until the end of the related module.

Sandwich Testing is an approach to combine top down testing with bottom up testing.

The main advantage of the Bottom-Up approach is that bugs are more easily found. With Top-Down, it is easier to find a missing branch link

5.6 Verification and validation

Verification and Validation are independent procedures that are used together for checking that a product, service, or system meets requirements and specifications and that it full fills its intended purpose. These are critical components of a quality management system such as ISO 9000. The words "verification" and "validation" are sometimes preceded with "Independent" (or IV&V), indicating that the verification and validation is to be performed by a disinterested third party.

The PMBOK guide, an IEEE standard, defines them as follows in its 4th edition

- "**Validation.** The assurance that a product, service, or system meets the needs of the customer and other identified stakeholders. It often involves acceptance and suitability with external customers. Contrast with *verification*."
- "**Verification**. The evaluation of whether or not a product, service, or system complies with a regulation, requirement, specification, or imposed condition. It is often an internal process. Contrast with *validation*."
    - Verification is intended to check that a product, service, or system (or portion thereof, or set thereof) meets a set of initial design specifications. In the development phase, verification procedures involve performing special tests to model or simulate a portion, or the entirety, of a product, service or system, then performing a review or analysis of the modelling results. In the post-development phase, verification procedures involve regularly repeating tests devised specifically to ensure that the product, service, or system continues to meet the initial design requirements, specifications, and regulations as time progresses.
    - Validation is intended to check that development and verification procedures for a product, service, or system (or portion thereof, or set thereof) result in a product,

service, or system (or portion thereof, or set thereof) that meets initial requirements.

- It is entirely possible that a product passes when verified but fails when validated. This can happen when, say, a product is built as per the specifications but the specifications themselves fail to address the user's needs.

System testing

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS). System testing tests not only the design, but also the behavior and even the believed expectations of the customer

Types of tests to include in system testing

The following examples are different types of testing that should be considered during System testing:

- Graphical user interface testing
- Usability testing
- Software performance testing
- Compatibility testing
- Exception handling
- Load testing
- Volume testing

- Stress testing
- Security testing
- Scalability testing
- Sanity testing
- Smoke testing
- Exploratory testing
- Ad hoc testing
- Regression testing
- Installation testing
- Maintenance testing Recovery testing and failover testing.
- Accessibility testing, including compliance with:
  - Americans with Disabilities Act of 1990
  - Section 508 Amendment to the Rehabilitation Act of 1973
  - Web Accessibility Initiative (WAI) of the World Wide Web Consortium (W3C)

Although different testing organizations may prescribe different tests as part of System testing, this list serves as a general framework or foundation to begin with.

Structure Testing:

It is concerned with exercising the internal logic of a program and traversing particular execution paths.

Output Testing:

- Output of test cases compared with the expected results created during design of test cases.
- Asking the user about the format required by them tests the output generated or displayed by the system under consideration.

- Here, the output format is considered into two ways, one is on screen and another one is printed format.

User acceptance Testing:

- Final Stage, before handling over to the customer which is usually carried out by the customer where the test cases are executed with actual data.
- It involves planning and execution of various types of test in order to demonstrate that the implemented software system satisfies the requirements stated in the requirement document.

Two set of acceptance test to be run:

- Those developed by quality assurance group.
- Those developed by customer

Chapter 6

CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

In conclusion, liver disease prediction using machine learning holds significant promise for improving early detection, prognosis, and patient management in clinical settings. Through the development and evaluation of predictive models leveraging advanced machine learning techniques, this study has contributed valuable insights into the field of liver disease diagnosis and prediction.

The findings of this study demonstrate the potential of machine learning models to accurately predict the presence or severity of liver disease based on patient data, including demographics, medical history, laboratory test results, and imaging studies. The predictive models developed in this study have shown promising performance in terms of accuracy, precision, recall, and other evaluation metrics, indicating their potential utility in real-world clinical practice.

Moreover, the application of Anaconda as a comprehensive platform for data analysis, model development, and deployment has facilitated the development and implementation of predictive models for liver disease prediction. The rich ecosystem of tools and libraries offered by Anaconda has enabled researchers and clinicians to explore innovative approaches to liver disease prediction and develop robust predictive models.

However, it is important to acknowledge the limitations and challenges associated with liver disease prediction using machine learning. These include the need for large and diverse datasets, potential biases in the data, interpretability of machine learning models, and generalizability to different patient populations and healthcare settings.

Moving forward, further research is warranted to address these challenges and enhance

the accuracy, reliability, and clinical utility of machine learning models for liver disease prediction. Future studies may focus on incorporating additional data sources, improving model interpretability, conducting external validation in diverse patient populations, and integrating predictive models into clinical workflows.

Overall, liver disease prediction using machine learning represents a promising approach to enhance clinical decision-making, improve patient outcomes, and advance our understanding of liver diseases. By leveraging the power of machine learning and Anaconda, we can continue to make strides in the field of liver disease diagnosis and prediction, ultimately benefiting patients and healthcare systems worldwide.

6.2 Future Enhancement

Future enhancements for liver disease prediction using machine learning can focus on several areas to further improve the accuracy, reliability, and clinical utility of predictive models. Here are some potential directions for future research and development:

Incorporation of Multi-Modal Data: Explore the integration of diverse data modalities, including genetic data, omics data (e.g., transcriptomics, metabolomics), and histopathological images, to capture a comprehensive view of liver disease pathophysiology and improve prediction accuracy.

Longitudinal Data Analysis: Investigate longitudinal patient data to model disease progression and monitor changes in liver health over time. Longitudinal models can provide insights into disease trajectories, treatment responses, and personalized patient management strategies.

Interpretability and Explainability: Develop interpretable machine learning models that provide insights into the underlying factors contributing to liver disease prediction. Incorporate techniques such as feature importance analysis, model visualization, and explanation methods to enhance model interpretability and trustworthiness.

Clinical Decision Support Systems: Integrate predictive models into clinical decision

support systems (CDSS) to assist healthcare professionals in making informed decisions about patient diagnosis, prognosis, and treatment planning. CDSS can provide real-time predictions, risk assessments, and treatment recommendations based on patient-specific data.

Personalized Medicine Approaches: Explore personalized medicine approaches to tailor predictive models to individual patient characteristics, including genetic predispositions, lifestyle factors, and comorbidities. Personalized models can improve prediction accuracy and enable precision medicine interventions.

Robustness and Generalizability: Enhance the robustness and generalizability of predictive models by addressing issues such as dataset biases, model overfitting, and domain adaptation. Incorporate techniques such as adversarial training, transfer learning, and data augmentation to improve model performance across diverse patient populations and healthcare settings.

Clinical Validation and Real-World Deployment: Conduct rigorous clinical validation studies to assess the performance and clinical utility of predictive models in real-world healthcare settings. Collaborate with healthcare institutions and regulatory agencies to ensure compliance with regulatory requirements and facilitate model deployment.

Ethical and Societal Implications: Consider the ethical, legal, and societal implications of deploying predictive models for liver disease prediction. Address issues such as patient privacy, data security, algorithmic bias, and healthcare disparities to promote equitable and responsible use of machine learning in healthcare.

ANNEXURE

Sample code

```
!pip show scikit-learn
!pip install --upgrade scikit-learn
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
# Load the dataset
data = pd.read_csv('liver_disease_dataset.csv')
# Preprocessing: handle missing values, encode categorical variables, etc.
# Split the dataset into features (X) and target variable (y)
X = data.drop('Liver_Disease', axis=1)
y = data['Liver_Disease'
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# Initialize the RandomForestClassifier
model = RandomForestClassifier()
# Train the model
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
```

```
print('Accuracy:', accuracy)
# Generate classification report
print('Classification Report:')
print(classification_report(y_test, y_pred))
# for numerical computing
import numpy as np
# for dataframes
import pandas as pd
# for easier visualization
import seaborn as sns
# for visualization and to display plots
from matplotlib import pyplot as plt
%matplotlib inline
# import color maps
from matplotlib.colors import ListedColormap
# Ignore Warnings
import warnings
warnings.filterwarnings("ignore")
from math import sqrt
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV

from sklearn.model_selection import cross_val_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
#from xgboost import XGBClassifier
```

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.metrics import roc_curve, auc, roc_auc_score, confusion_matrix
from sklearn.preprocessing import StandardScaler
#from sklearn.cross_validation import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.neighbors import KNeighborsClassifier
from matplotlib.colors import ListedColormap
from sklearn.metrics import accuracy_score
df=pd.read_csv('indian_liver_patient.csv')
df.shape
df.columns
df.head()
df.dtypes[df.dtypes=='object']
# Plot histogram grid
df.hist(figsize=(15,15), xrot=-45, bins=10) ## Display the labels rotated by 45 degress
# Clear the text "residue"
plt.show()
df.describe()
## if score==negative, mark 0 ;else 1
def partition(x):
    if x == 2:
  return 0
  return 1
df['Dataset'] = df['Dataset'].map(partition)
df.describe(include=['object'])
plt.figure(figsize=(5,5))
```

```
sns.countplot(y='Gender', data=df)
# Create separate object for target variable
y = df.Dataset
```

Data Preparation

```
# Create separate object for input features
X = df.drop('Dataset', axis=1)
# Split X and y into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    test_size=0.2,
                                                    random_state=1234,
                                                    stratify=df.Dataset)
# Print number of observations in X_train, X_test, y_train, and y_test
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
train_mean = X_train.mean()
train_std = X_train.std()
## Standardize the train data set
X_train = (X_train - train_mean) / train_std
## Check for mean and std dev.
X_train.describe()
## Note: We use train_mean and train_std_dev to standardize test data set
X_test = (X_test - train_mean) / train_std
## Check for mean and std dev. - not exactly 0 and 1
X_test.describe()
```

Model-1 Logistic Regression

```
tuned_params = {'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'penalty': ['l1',
'l2']}
model = GridSearchCV(LogisticRegression(), tuned_params, scoring = 'roc_auc',
```

```
n_jobs=-1)
model.fit(X_train, y_train)
model.best_estimator_
## Predict Train set results
y_train_pred = model.predict(X_train)
## Predict Test set results
y_pred = model.predict(X_test)
# Get just the prediction for the positive class (1)
y_pred_proba = model.predict_proba(X_test)[:,1]
# Display first 10 predictions
y_pred_proba[:10]
# Display first 10 predictions
y_pred_proba[:10]
i=28  ## Change the value of i to get the details of any point (56, 213, etc.)
print('For test point {}, actual class = {}, precited class = {}, predicted probability = {}'.
    format(i, y_test.iloc[i], y_pred[i], y_pred_proba[i]))
confusion_matrix(y_test, y_pred).T
# Calculate ROC curve from y_test and pred
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
# Plot the ROC curve
fig = plt.figure(figsize=(8,8))
plt.title('Receiver Operating Characteristic')
# Plot ROC curve
plt.plot(fpr, tpr, label='l1')
plt.legend(loc='lower right')
# Diagonal 45 degree line
plt.plot([0,1],[0,1],'k--')
```

```
# Axes limits and labels
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
# Calculate AUC for Train set
print(roc_auc_score(y_train, y_train_pred))
# Calculate AUC for Test set
print(auc(fpr, tpr))
## Building the model again with the best hyperparameters
model = LogisticRegression(C=1, penalty = 'l2')
model.fit(X_train, y_train)
indices = np.argsort(-abs(model.coef_[0,:]))
print("The features in order of importance are:")
print(50*'-')
for feature in X.columns[indices]:
    print(feature)
```

Model-2 Random Forest

```
tuned_params = {'C': [0.0001, 0.001, 0.01, 0.1, 1, 10, 100, 1000, 10000], 'penalty': ['l1', 'l2']}
model = GridSearchCV(LogisticRegression(), tuned_params, scoring = 'roc_auc', n_jobs=-1)
model.fit(X_train, y_train)
model.best_estimator_
```

```
## Predict Train set results
y_train_pred = model.predict(X_train)
## Predict Test set results
y_pred = model.predict(X_test)
# Get just the prediction for the positive class (1)
y_pred_proba = model.predict_proba(X_test)[:,1]
# Display first 10 predictions
y_pred_proba[:10]
i=28  ## Change the value of i to get the details of any point (56, 213, etc.)
print('For test point {}, actual class = {}, precited class = {}, predicted probability = {}'.
     format(i, y_test.iloc[i], y_pred[i], y_pred_proba[i]))
confusion_matrix(y_test, y_pred).T
# Calculate ROC curve from y_test and pred
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
# Plot the ROC curve
fig = plt.figure(figsize=(8,8))
plt.title('Receiver Operating Characteristic')

# Plot ROC curve
plt.plot(fpr, tpr, label='l1')
plt.legend(loc='lower right')
# Diagonal 45 degree line
plt.plot([0,1],[0,1],'k--')
# Axes limits and labels
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
```

```
plt.xlabel('False Positive Rate')
plt.show()
# Calculate AUC for Train set
print(roc_auc_score(y_train, y_train_pred))
# Calculate AUC for Test set
print(auc(fpr, tpr))
## Building the model again with the best hyperparameters
model = LogisticRegression(C=1, penalty = 'l2')
model.fit(X_train, y_train)
indices = np.argsort(-abs(model.coef_[0,:]))
print("The features in order of importance are:")
print(50*'-')
for feature in X.columns[indices]:
    print(feature)
```

Model-3 Descision Trees

```
tuned_params = {'n_estimators': [100, 200, 300, 400, 500], 'min_samples_split': [2, 5, 10], 'min_samples_leaf': [1, 2, 4]}
model = RandomizedSearchCV(RandomForestClassifier(), tuned_params, n_iter=15, scoring = 'roc_auc', n_jobs=-1)
model.fit(X_train, y_train)
model.best_estimator_
y_train_pred = model.predict(X_train)
y_pred = model.predict(X_test)
y_pred_proba = model.predict_proba(X_test)[:,1]
y_pred_proba[:10]
confusion_matrix(y_test, y_pred).T
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
```

```
# Plot the ROC curve
fig = plt.figure(figsize=(8,8))
plt.title('Receiver Operating Characteristic')
# Plot ROC curve
plt.plot(fpr, tpr, label='l1')
plt.legend(loc='lower right')
# Diagonal 45 degree line
plt.plot([0,1],[0,1],'k--')
# Axes limits and labels
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
# Calculate AUC for Train
roc_auc_score(y_train, y_train_pred)
print(auc(fpr, tpr))
## Building the model again with the best hyperparameters
model = DecisionTreeClassifier(min_samples_split=2, min_samples_leaf=6,
max_depth=4)
model.fit(X_train, y_train)
indices = np.argsort(-model.feature_importances_)
print("The features in order of importance are:")
print(50*'-')
for feature in X.columns[indices]:
    print(feature)
```

Model-4 Gradient Boosting

```
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report,confusion_matrix
from sklearn.ensemble import AdaBoostClassifier, BaggingClassifier
from sklearn.linear_model import Perceptron
from sklearn.linear_model import SGDClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.ensemble import GradientBoostingClassifier
# Instantiate the GradientBoostingClassifier with default parameters
gbclass = GradientBoostingClassifier(random_state=1000, verbose=0, max_depth=3)
# Fit the model with training data
gbclass.fit(X_train, y_train)
# Predict Output
predicted = gbclass.predict(X_test)
## Predict Train results
y_train_pred = gbclass.predict(X_train)
## Predict Test results
y_pred = gbclass.predict(X_test) from sklearn.ensemble import
GradientBoostingClassifier
# Instantiate the GradientBoostingClassifier
gbclass = GradientBoostingClassifier()
# Fit the model with training data
gbclass.fit(X_train, y_train)
# Now you can use predict_proba method
y_pred_proba = gbclass.predict_proba(X_test)[:, 1]
# Calculate ROC curve from y_test and pred
```

```
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
 # Plot the ROC curve
fig = plt.figure(figsize=(8,8))
plt.title('Receiver Operating Characteristic')
# Plot ROC curve
plt.plot(fpr, tpr, label='l1')
plt.legend(loc='lower right')
# Diagonal 45 degree line
plt.plot([0,1],[0,1],'k--')
# Axes limits and labels
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
roc_auc_score(y_train,y_train_pred )
# Calculate AUC for Test
print(auc(fpr, tpr))
# Neural Networks# Neural
```

Model - 5 Neural Networks

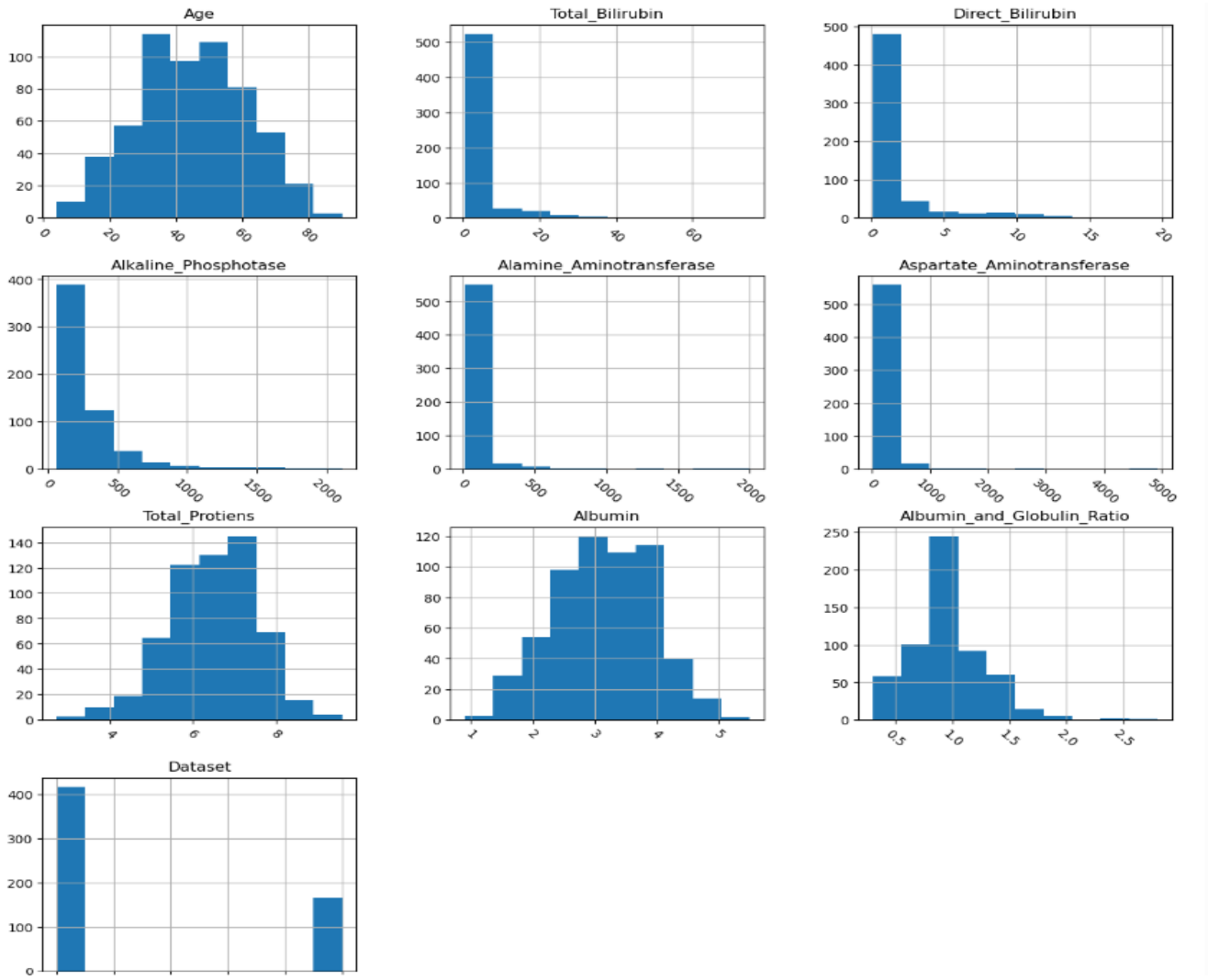
```
neural = MLPClassifier(hidden_layer_sizes=40,
                       activation='relu',
                       solver='adam',
                       alpha=0.001,
                       batch_size='auto',
                       max_iter=200,
                       random_state=137,
```

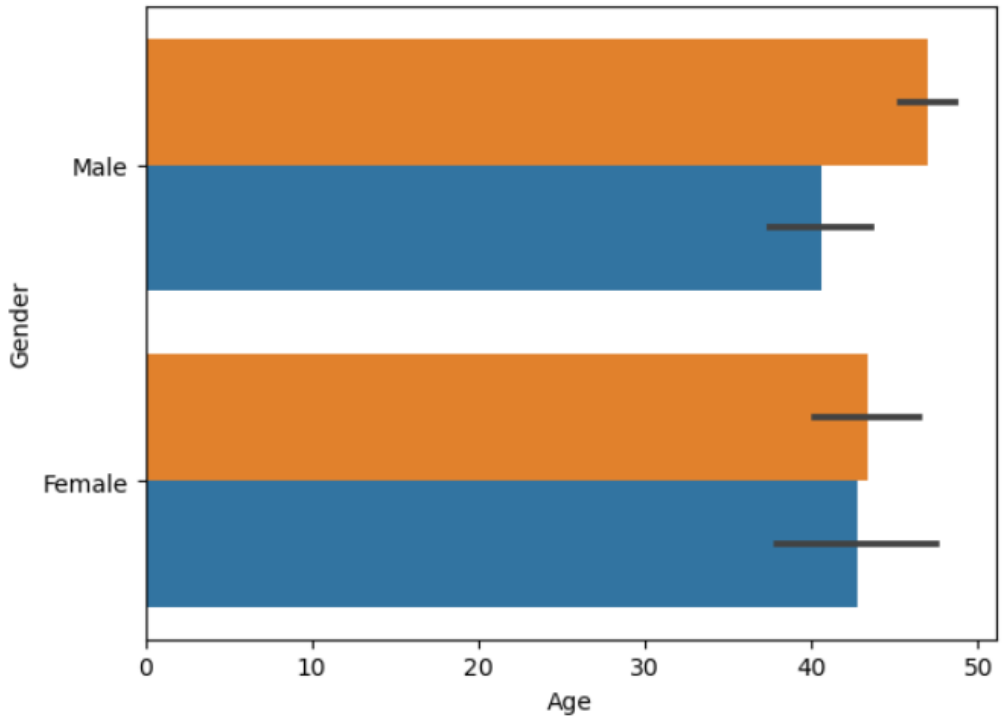
```
                tol=0.0001,
                early_stopping=False,
                validation_fraction=0.1,
                beta_1=0.9,
                beta_2=0.999,
                epsilon=1e-08,
                learning_rate='constant',
                power_t=0.5,
                momentum=0.8,
                nesterovs_momentum=True,
                shuffle=True,
                learning_rate_init=0.001)
neural.fit(X_train, y_train)
#Predict Output
predicted = neural.predict(X_test)
neural_score = round(neural.score(X_train, y_train) * 100, 2)
neural_score_test = round(neural.score(X_test, y_test) * 100, 2)
print('Neural Score: \n', neural_score)
print('Neural Test Score: \n', neural_score_test)
print('Accuracy: \n', accuracy_score(y_test, predicted))
print(confusion_matrix(predicted,y_test))
print(classification_report(y_test,predicted))
## Predict Train results
y_train_pred = neural.predict(X_train)
## Predict Test results
y_pred = neural.predict(X_test)
y_pred_proba = neural.predict_proba(X_test)[:,1]
```

```
# Calculate ROC curve from y_test and pred
fpr, tpr, thresholds = roc_curve(y_test, y_pred_proba)
# Plot the ROC curve
fig = plt.figure(figsize=(8,8))
plt.title('Receiver Operating Characteristic')
# Plot ROC curve
plt.plot(fpr, tpr, label='l1')
plt.legend(loc='lower right')
# Diagonal 45 degree line
plt.plot([0,1],[0,1],'k--')
#Axes limits and labels
plt.xlim([-0.1,1.1])
plt.ylim([-0.1,1.1])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
roc_auc_score(y_train,y_train_pred )
# Calculate AUC for Test
print(auc(fpr, tpr))
# Example test vector
test_vector = np.array([40, 1, 1.5, 0.8, 300, 35, 40, 7.2, 3.9, 1.1])
# Convert the Series object to a numpy array
test_vector_preprocessed_np = test_vector_preprocessed.to_numpy()
# Reshape the numpy array to a 2D array with one row
test_vector_preprocessed_reshaped = test_vector_preprocessed_np.reshape(1, -1)
# Use the trained model to predict the target value
predicted_y = model.predict(test_vector_preprocessed_reshaped)
```

```
# Print the predicted target value
print("Predicted target value:", predicted_y[0])
```

Sample Output





Age seems to be a factor for liver disease for both male and female genders

Reference:

- [1] Upadhyay, Y. and Wasson, V. 2014. "Analysis of Liver MR Images for Cancer Detection using Genetic Algorithm". International Journal of Engineering Research and General Science. Vol.2, No.4, PP: 730-737.
- [2] Kumar, P. Bhalerao, S. 2014. "Detection of Tumor in Liver Using Image Segmentation and Registration Technique". IOSR Journal of Electronics and Communication Engineering (IOSR-JECE). Vo.9, No.2, PP: 110-115.

- [3] Selle, D.; Spindler, W.; Preim, B. and Peitgen, H. O. 2000. "Mathematical Methods in Medical Imaging: Analysis of Vascular Structures for Liver Surgery Planning". PP: 1-21.
- [4] Zimmer, C. and Olivo-Marin, J. C. 2005. "Coupled Parametric Active Contours". Transactions on pattern Analysis and Machine Intelligence. Vol.27, No.11, PP: 1838-1841.
- [5] Chitra, S. and Balakrishnan, G. 2012. "Comparative Study for Two Color Spaces HSCbCr and YCbCr in Skin Color Detection". Applied Mathematical Sciences. Vol.6, No.85, PP: 4229 – 4238.
- [6] M. Lalonde, M. Beaulieu, and L. Gagnon, "Fast and robust optic disc detection using pyramidal decomposition and Hausdorff-based template matching," *IEEE Trans. Med. Imag.*, vol. 20, no. 11, pp. 1193–1200, Nov. 2001
- [7] T.Kauppi and H.Kälviäinen, "Simple and robust optic disc localization using colour decorrelated templates," in *Proc. 10th Int. Conf. Advanced Concepts for Intell. Vision Syst.,*. Berlin, Germany: Springer-Verlag, 2008
- [8] Leida Li, Shushang Li, Hancheng Zhu, "An efficient scheme for detecting Copy-Move forged images by local binary patterns", Journal of Information Hiding and Multimedia Signal Processing, Vol. 4, No. 1, pp. 46-56, January 2013.
- [9] Hailing Huang, Weiqiang Guo, Yu Zhang, "Detection of Copy-Move Forgery in Digital Images Using SIFT Algorithm", IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application, Wuhan, China, 2008.
- [10] Y.Sutcu, B.Coskun, H.T.Sencar, N.Memmon, "Tamper detection based on regularity of wavelet transform coefficients", IEEE International Conference on Image Processing, 2007