DISEASE DIAGNOSIS USING MACHINE LEARNING

The project report submitted to
Guru Jambheshwar University of Science and Technology, Hisar
for the partial award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

by

Nishu
(200010130082)

Dr. Jyoti, Prof.
Department of CSE

Department of Computer Science & Engineering

GURU JAMBHESHWAR UNIVERSITY OF SCIENCE AND TECHNOLOGY, HISAR

June, 2024

DECLARATION

I, Nishu, 200010130082, certify that the work contained in this project report is original and was performed by me under the direction of my supervisor. This work has not been submitted to any other institution for the awarding of a degree, and I followed the ethical practices and other guidelines of the Department of Computer Science and Engineering in preparing my report. Whenever I use material (data, theoretical analysis, figures, text) from other sources, I cite them in the body of the report and give details in the references.

Signature

Nishu
200010130082
Department of Computer Science and Engineering
Guru Jambheshwar University of Science and Technology, Hisar

Signature

Dr. Jyoti, Professor
Department of Computer Science and Engineering
Guru Jambheshwar University of Science and Technology, Hisar

ACKNOWLEDGEMENT

I would like to express my special thanks to my teacher who helped me to complete this project and also helped me with a lot of research, so I learned many new things. Secondly, I would also like to thank my parents, friends and classmates who helped me a lot to complete this project within the limited time frame. Last but not least, I would like to thank everyone who directly or indirectly supported me in completing this project.

CERTIFICATE

This is to certify that Nishu has roll no. 200010130082 is a student of B.Tech (CSE-2), Department of Computer Science and Engineering, Guru Jambheshwar University of Science and Technology, Hisar has completed the project entitled “Disease Diagnosis Using Machine Learning”.

Dr. Jyoti, Professor
Department of CSE
GJUS&T Hisar

Abstract

This project uses a comparative analysis of machine learning (ML) algorithms that enhance disease diagnosis. Traditional diagnosis methods are time-consuming and error-prone. ML techniques are used to improve the accuracy and efficiency of prediction. In the project, datasets are acquired from Kaggle and performed preprocessing, feature engineering, and model training. Various classification algorithms are used, and model performance is evaluated. The prediction results of different ML models are concluded to find the best out of all models. The findings indicate that ML models improve diagnostic speed and reliability, offering scalable solutions for underserved populations and contributing to precision medicine and healthcare.

LIST OF FIGURES

CONTENTS

CHAPTER 1

Introduction

1.1 Introduction to Project:

Diagnosing diseases is an essential component of providing quality healthcare. It entails the comprehensive analysis and comprehension of a patient's ailment utilizing a combination of their reported symptoms, detailed clinical history, and results from various diagnostic tests. Traditionally, skilled healthcare professionals have played a pivotal role in interpreting and analyzing patient symptoms, medical backgrounds, and test findings to arrive at accurate diagnoses. Nevertheless, this process can often be time-intensive, influenced by subjective judgment, and susceptible to human errors, particularly when dealing with intricate or uncommon diseases that manifest with subtle or atypical symptoms. or rare diseases with subtle or unusual symptoms.

In the traditional approach to healthcare, medical professionals rely on their knowledge and experience to interpret a patient's medical history and symptoms. This process usually begins with a comprehensive interview and physical examination to gather a complete medical background. Subsequently, physicians may request further diagnostic tests, including blood tests, imaging studies (such as X-rays, MRIs, or CT scans), biopsies, and other laboratory tests based on the initial assessment. The accurate interpretation of these tests relies on a deep understanding of medicine and human biology. Physicians utilize their expertise to recognize patterns in symptoms and test results that may point to specific diseases or conditions. However, due to the subjective nature of this process, finding an accurate diagnosis can be particularly challenging, especially in cases involving rare diseases or scenarios where symptoms are similar or ambiguous.

With the use of traditional methods, several methods contribute to challenges faced in Disease Diagnosis. Like:

- The complexity of symptoms in various diseases is notable. It's important to be aware that many illnesses share common symptoms. For instance, symptoms like fatigue, fever, and weight loss can be indicative of a wide range of conditions, from common infections to more serious chronic diseases such as cancer. Therefore, it's essential to consider the broader clinical context and seek medical advice for proper evaluation and diagnosis when experiencing such symptoms.
- The diagnostic process is prone to human error due to a variety of reasons. Physicians may inadvertently overlook subtle symptoms, misinterpret test results, or make incorrect assumptions based on incomplete information. Additionally, factors such as fatigue, stress, or distractions can further contribute to the possibility of human error in the diagnostic process. Healthcare professionals must remain vigilant and adopt thorough, systematic approaches to minimize the impact of human error in medical diagnoses.

To overcome these types of challenges Machine Learning comes into the role, to make the diagnosis faster and more accurate. Machine learning techniques have the potential to transform disease diagnosis by harnessing the power of data analysis and pattern recognition. These algorithms learn from large volumes of data to identify complex patterns, correlations, and predictive features that might not be immediately apparent to humans.

The use of predetermined mathematical functions yields a result (classification or regression) that is frequently difficult for humans to accomplish. Machine learning has emerged as a powerful tool in healthcare, offering new avenues for improving disease diagnosis through data analysis and pattern recognition. These algorithms learn from large volumes of data to identify complex patterns, correlations, and predictive features that might not be immediately apparent

to humans. By leveraging this technology, healthcare systems can improve diagnostic accuracy, speed, and efficiency.

- Machine learning algorithms can swiftly process and analyze large datasets, enabling faster diagnoses and timely treatment when compared to human capacities.
- Machine learning models can attain high levels of accuracy in disease diagnosis by analyzing extensive data. They are capable of detecting nuanced patterns and correlations that may escape human doctors' notice.
- Machine learning models offer consistency that is free from the impact of fatigue or cognitive biases experienced by humans. This consistency results in more reliable and accurate diagnostic outcomes, minimizing the probability of errors.
- Scalability is a crucial advantage of machine learning models, as it allows for their widespread deployment, making it possible to offer diagnostic services to underserved populations and regions with a scarcity of healthcare professionals. This scalability ensures that individuals in remote or underprivileged areas have access to essential diagnostic support without needing to travel long distances to urban centres.

1.2 Motivation:

The motivation behind evaluating various prediction algorithms for disease diagnosis in a machine-learning model stems from the pressing need to enhance diagnostic accuracy, efficiency, and reliability in healthcare. Different algorithms come with unique strengths and capabilities, such as their ability to handle varying data complexities, achieve diverse levels of precision, and optimize computational efficiency. Through systematic comparisons of these algorithms, healthcare practitioners and researchers strive to identify the most effective algorithm for specific conditions, taking into account factors such as the type of disease, the nature of the data (structured vs. unstructured), and the clinical context.

This rigorous evaluation aids in pinpointing the most suitable algorithm that can deliver robust, accurate, and timely diagnoses, consequently leading to improved patient outcomes. Moreover, the careful selection of the best algorithm minimizes errors and biases, ensuring that the diagnostic process remains as objective and evidence-based as possible. This approach also contributes to the broader goal of advancing precision medicine by upholding the credibility and reliability of diagnostic tools. This field aims to tailor treatments and interventions to individual patient profiles based on the most accurate diagnostic information available, ultimately promoting personalized and effective healthcare delivery.

A few major points for using different machine learning algorithms to predict disease diagnosis are as follows:

- In medical diagnostics, different machine learning algorithms have varying strengths and weaknesses depending on the specific disease and data characteristics. Healthcare professionals can identify the most accurate diagnostic approach by evaluating and comparing multiple algorithms, minimizing false results and ensuring precise diagnoses and suitable treatment.
- Healthcare data includes structured data like patient demographics and lab results, unstructured data like clinical notes, and imaging data such as X-rays and MRIs. Different machine learning algorithms have varying effectiveness in handling these diverse data types. By evaluating multiple algorithms, it's possible to identify the most suitable one for each specific data type or combination, leading to more comprehensive and accurate diagnoses

CHAPTER 2

Background & Literature Review:

2.1 Background:

The intersection of machine learning and healthcare has witnessed significant strides in recent years, particularly in the domain of multiple disease diagnosis. The traditional diagnostic paradigm, heavily reliant on manual analysis and often prone to human error, faces challenges in efficiently handling complex scenarios involving multiple coexisting conditions. Machine learning, with its data-driven and pattern recognition capabilities, emerges as a promising solution to enhance diagnostic accuracy, speed, and comprehensiveness in the context of multiple diseases.

The prevalence of comorbidities, where individuals experience two or more coexisting health conditions, poses a substantial burden on healthcare systems globally. Accurate and timely diagnosis becomes paramount in managing such complex cases, necessitating advanced tools that can navigate through intricate datasets and discern patterns indicative of multiple diseases simultaneously.

2.2 Literature Review:

1. Purushottam et. al. [1]- Hill climbing and decision tree algorithms were proposed in a study by. are used in the System for Effective Heart Disease Prediction. The outcomes of algorithms like SVM and KNN are based on split conditions that can be vertical or horizontal depending on the dependent variables. Yet, a decision tree is a structure that resembles a tree with a root node, leaves, and branches, and it is based on the decisions made in each tree. The value of the attributes in the dataset is also explained by the decision tree. Also, they used the Cleveland data set. The accuracy of this method is 91%.
   Naive Bayes, the second algorithm, is used for categorization.

2. Mariam et. al. [2]- studied compared the performance of two different classifiers, Naive Bayes and K Nearest Neighbors (KNN), for classifying breast cancer. After conducting cross-validation, the KNN classifier achieved an accuracy of 97.51%, which was the highest among the two classifiers and had the lowest error rate. In comparison, the Naive Bayes classifier achieved an accuracy of 96.19%. This indicates that KNN outperformed Naive Bayes in accurately classifying breast cancer in this particular study.

3. Ankita Tyagi and Rikitha Mehra et. al. [3]– In their project of "Interactive Thyroid Disease Prediction System Using Machine Learning Techniques", they use different classification algorithms- Decision Tree, Support Vector Machine, Artificial Neural Network, k-Nearer-Neighbor algorithm. Based on the data set obtained from UCI Repository, classification and prediction was performed and accuracy was obtained based on output produced. They have analyzed accuracy of algorithms used and comparison is made to find best technique with high accuracy.

4. Chaurasia et al. [4]- conducted an in-depth analysis to evaluate the performance of various supervised learning classifiers using a dataset related to the growth of Wisconsin breast cancer. I applied different machine learning techniques including Naive Bayes, Support Vector Machine, Neural Networks, and Decision Tree. Upon analyzing the results, it was found that the Support Vector Machine yielded the most precise outcome with an impressive accuracy score of 96.84%.

5. Avinash Golande et. al. [5]- proposed that several data mining techniques are utilized in "Heart Disease Prediction Using Efficient Machine Learning Approaches," which helps doctors distinguish between different types of heart disease. K-Nearest Neighbor, Decision Tree, and Naive Bayes are common techniques. Packing calculation, part thickness, consecutive negligible streamlining, neural systems, straight kernel selfarranging guidance, and SVM are other novel characterization-based procedures that are used (Bolster Vector Machine).

2.3 Existing systems:

There are several existing models and research papers that focus on the comparative analysis of machine learning algorithms for disease diagnosis. Here are details about two notable studies:

1. **Comparative Analysis of Machine Learning Models for Chronic Kidney Disease Prediction**In this study, various machine learning models were compared for their ability to predict chronic kidney disease (CKD). The findings revealed that the support vector machine with the Laplace kernel function demonstrated superior performance in terms of classification accuracy, while the random forest model also exhibited competitive results. These findings underscore the efficacy of machine learning techniques in enhancing the accuracy of CKD diagnosis, a critical factor for early intervention and treatment.

2. **Comparative Analysis of Explainable Machine Learning Prediction Models for Hospital Mortality**: This research focuses on predicting hospital mortality using different machine learning models. The study evaluates Random Forest (RF), Logistic Regression (LR), Adaptive Boost Classifier (ADA), and Naive Bayes (NB) models. The models were trained and validated on a dataset containing patient records from an Intensive Care Unit (ICU). The performance was measured using the area under the receiver operating characteristic curve (AUC), with the Random Forest model achieving the highest AUC, indicating the best performance in predicting hospital mortality. The study also explores the use of SHAP values for model interpretability, helping to understand the contribution of different features to the predictions.

3. **Comparative Analysis of Machine Learning Algorithms for Multi-Syndrome Classification of Neurodegenerative Syndromes**: This paper evaluates machine learning

models for classifying different neurodegenerative syndromes, including Alzheimer's disease, Parkinson's disease, and various forms of primary progressive aphasia. The models analyzed include Random Forest (RF), Gradient Boosting (GB), and Support Vector Machine (SVM) with various kernel functions. The study employed a 5fold cross-validation method and optimized the models using Bayesian optimization. The results indicated that the SVM with a linear kernel provided the best performance for this multi-syndrome classification task.

CHAPTER 3

PROBLEM FORMULATION AND OBJECTIVE

3.1 Problem Statement :

This research project aims to develop a robust disease prediction model using machine learning and perform a comparative analysis of different machine learning algorithms to identify the most accurate and efficient model for predicting various diseases. By leveraging four distinct datasets corresponding to different medical conditions, the study will evaluate the performance of algorithms such as Decision Trees, Random Forests, Support Vector Machines (SVM) and Logistic Regression. Each algorithm will be assessed based on metrics like accuracy, precision, recall, F1 score, and computational efficiency.

- As objective of this project, is to create a Machine Learning model for Disease Diagnosis with comparative analysis of different machine-learning algorithms.

CHAPTER 4

SOFTWARE REQUIREMENTS SPECIFICATIONS

4.1 SOFTWARE REQUIREMENTS:

4.1.1 Operating System: Windows7/8/10

4.1.2 Coding Language: Python

4.1.3 Anaconda Distribution

Anaconda is a comprehensive data science platform that includes a collection of tools and libraries essential for data analysis, scientific computing, and machine learning. It simplifies package management and deployment, making it easier to work with Jupyter Notebook and various machine-learning frameworks.

- **Package Management**: Anaconda comes with Conda, a package manager that simplifies the installation, updating, and removal of software packages.

- **Virtual Environments**: Allows the creation of isolated environments to manage dependencies for different projects.

4.1.4 Jupyter:

Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. It is widely used for data

cleaning and transformation, numerical simulation, statistical modelling, data visualization, machine learning, and much more.

- **Cells**: The fundamental unit of a notebook, which can contain code, text, equations, or visualizations.
- **Kernel**: Executes the code contained in the notebook cells.
- **Interactive Widgets**: Facilitate the creation of interactive controls for real-time data manipulation and visualization.

1. Code Editor

Jupyter Notebook, included in the Anaconda distribution, serves as a versatile and interactive code editor ideal for developing machine learning models. Key features include:

- **Interactive Coding Environment**: Facilitates code writing and execution in cells for easy testing and iteration of machine learning algorithms.
- **Support for Multiple Languages**: Supports Python, integrates with other languages like R, Julia, and SQL.
- **Markdown Support**: Allows for documentation to be included with code, which is important for describing the steps and processes in the notebook.

2. Debug and Test

Jupyter Notebook facilitates debugging and testing of machine learning models through:

- **Inline Debugging**: Code cells can be executed individually, allowing for immediate feedback and troubleshooting.
- **Magic Commands**: %debug, %timeit, and %run are examples of magic commands that help in profiling and debugging code.
- **Visualization**: Inline plotting with libraries such as Matplotlib and Seaborn helps in visualizing data and model performance.

3. Project Management

Managing a machine learning project for disease diagnosis in Jupyter Notebook involves:

- **Notebook Organization**: Notebooks can be organized into directories and subdirectories within the Anaconda environment, facilitating modular project structure.
- **Version Control Integration**: Integration with version control systems like Git enables tracking changes and collaboration among multiple developers.
- **Environment Management**: Anaconda's Conda package manager allows creating isolated environments with specific dependencies, ensuring consistency across different stages of the project.

4. Data Preprocessing and Analysis

Jupyter Notebook supports extensive data preprocessing and analysis capabilities, which are crucial for machine learning projects:

- **Pandas Library**: For data manipulation and analysis, including operations like merging, reshaping, selecting, and cleaning data.
- **NumPy Library**: For numerical computing and handling arrays, which is essential for data preparation and feature engineering.
- **Scikit-learn Library**: Provides a range of preprocessing tools such as scaling, encoding, and transformation, which are critical steps in preparing data for machine learning models.

5. Model Evaluation

Evaluation of machine learning models in Jupyter Notebook is facilitated by:

- **Metrics and Scoring**: Functions for calculating accuracy, precision, recall, F1 score, and AUCROC, which are critical for assessing model performance.
- **Cross-Validation**: Methods for validating models through techniques like k-fold cross-validation to ensure generalizability.
- **Visualization Tools**: Libraries such as Matplotlib, Seaborn, and Plotly for creating detailed plots and visual representations of model performance metrics.

6. Deployment and Reporting

Jupyter Notebook supports deployment and reporting functionalities such as:

- **Exporting Notebooks**: The capability to export notebooks into different formats such as HTML, PDF, and Markdown for the purpose of reporting and presenting.
- **Integration with Web Frameworks**: Tools like Flask or Django for deploying machine learning models as web services.
- **Interactive Widgets**: Ipywidgets allows for interactive controls within the notebook, improving the user experience for reporting and analysis.

4.2 HARDWARE REQUIREMENTS

4.2.1 Processor: Any Processor above 500 MHz.

4.2.2 Ram: 4 GB

4.2.3 Hard Disk: 250 GB

4.2.4 Input device: Standard Keyboard and Mouse

4.2.5 Output device: High Resolution Monitor.

CHAPTER 5

METHODOLOGY AND TOOLS USED

5.1 Methodology:

A disease diagnosis using comparative analysis of different machine learning prediction algorithms includes the following methodologies and tools:

5.1.1 Collection of disease datasets:

- **Sources**: Collect datasets from reliable sources such as medical records, public health databases, and research repositories. Ensure the datasets cover a variety of diseases to provide a comprehensive analysis.
- **Datasets**: For this project, we use four different datasets corresponding to diseases such as diabetes, heart disease, lung cancer, and kidney disease.

5.1.2 Data Preprocessing:

- **Cleaning**: Handle missing values, remove duplicates, and correct errors in the datasets.
- **Normalization**: Normalize the data to ensure all features are on a similar scale, which is crucial for algorithms like SVM and neural networks.
- **Feature Engineering**: Create new features based on domain knowledge to enhance model performance.
- **Encoding**: Convert categorical variables into numerical values using techniques such as one-hot encoding.

5.1.3. Exploratory Data Analysis (EDA)

- **Descriptive Statistics**: Calculate mean, median, mode, standard deviation, etc., to understand the data distribution.
- **Visualization**: Use libraries like Matplotlib and Seaborn, to visualize the data. Create histograms, box plots, scatter plots, and correlation matrices to identify patterns and relationships between features.

5.1.4. Model Selection and **Implementation:**

Choose a diverse set of machine learning algorithms to ensure a comprehensive comparison:

- **Logistic Regression:**

  Logistic Regression is a fundamental and widely used statistical method in machine learning for binary classification tasks. It is particularly effective for disease diagnosis due to its simplicity, interpretability, and effectiveness in handling binary outcomes, such as the presence or absence of a disease. Logistic Regression models the probability that a given input belongs to a particular class. Unlike linear regression, which predicts a continuous outcome, logistic regression predicts a binary outcome by estimating the probability that an observation falls into one of two categories.

  Model Formulation:

  Logistic regression estimates the probability P(Y=1|X)P(Y=1|X)P(Y=1|X) where Y is the target variable (e.g., disease presence: 1 for positive, 0 for negative) and XXX is the vector of input features. The model is formulated as:

$$P(Y = 1|X) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 X_1 + \beta_2 X_2 + \ldots + \beta_n X_n)}}$$

where β_0 is the intercept and $\beta_1, \beta_2, \ldots, \beta_n$ are the coefficients for the input features.

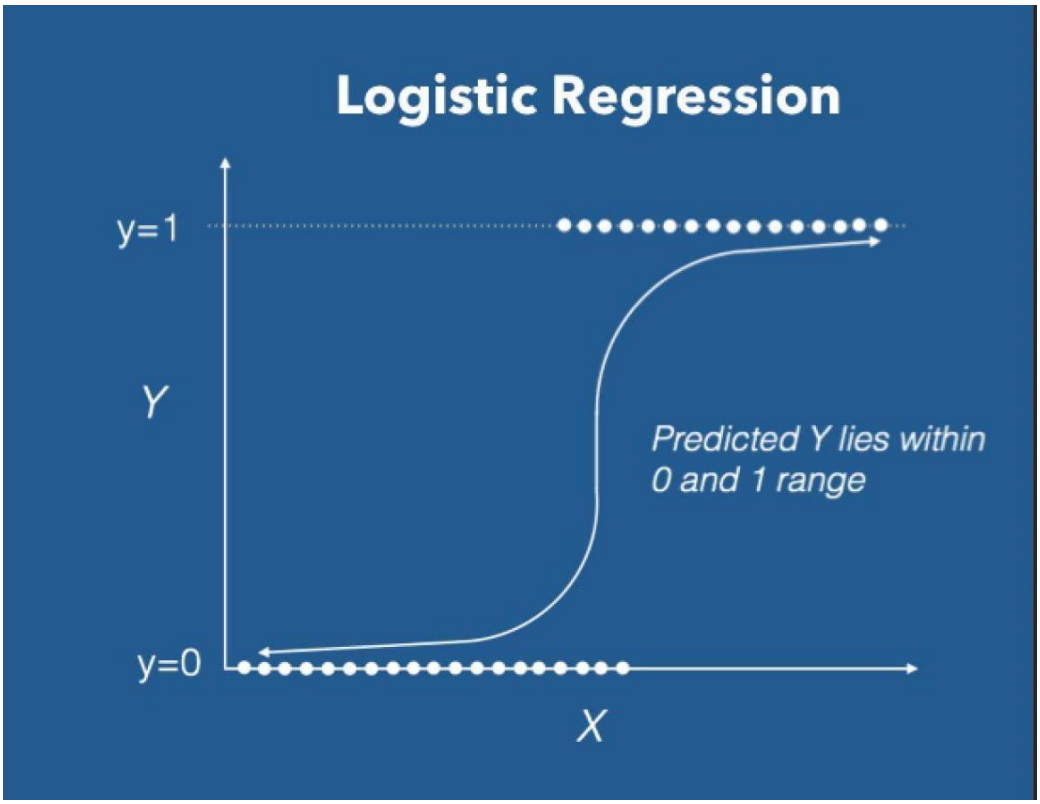

Fig. 5.1 – Logistic Regression

- **Decision Trees:**

  Decision tree classification is a popular machine-learning technique used for both classification and regression tasks. It is especially useful in the field of disease diagnosis, where the goal is to predict the presence or absence of a disease based on various patient features. In this project, we employed decision tree classification to diagnose diabetes, using a dataset that includes several health metrics of patients. A decision tree is a flowchart-like structure where an internal node represents a feature (or attribute), the branch represents a decision rule, and each leaf node represents the outcome. The paths from root to leaf represent classification rules.

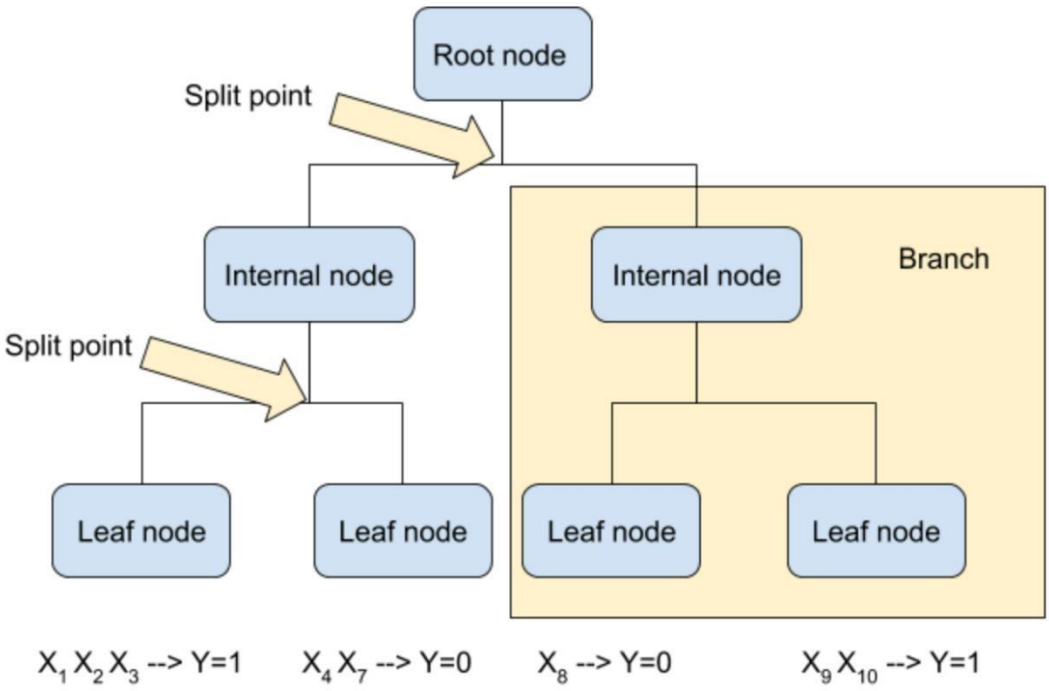

Fig. 5.2 – Decision Tree Classifier

- **Random Forests:**

Random Forest is a versatile and powerful machine-learning algorithm capable of performing both classification and regression tasks. It is particularly well-suited for complex tasks such as disease diagnosis due to its robustness, accuracy, and ability to handle high-dimensional data. This section details the workings of the Random Forest Classifier and its application in predicting diseases within the context of machine learning.

Random Forest is an ensemble learning method, meaning it combines the predictions of multiple base estimators to improve accuracy and control overfitting. It builds multiple decision trees during training and outputs the mode of the classes (classification) or mean prediction (regression) of the individual trees.

How Random Forest Works:

1. **Dataset Preparation**: The dataset is divided into several subsets using bootstrap sampling.

2. **Decision Tree Construction**: For each subset, a decision tree is constructed by selecting random subsets of features at each node. This introduces diversity among the trees.
3. **Voting/Averaging**: In classification, each tree votes for a class, and the class with the majority vote is chosen. In regression, the average of all tree predictions is taken.

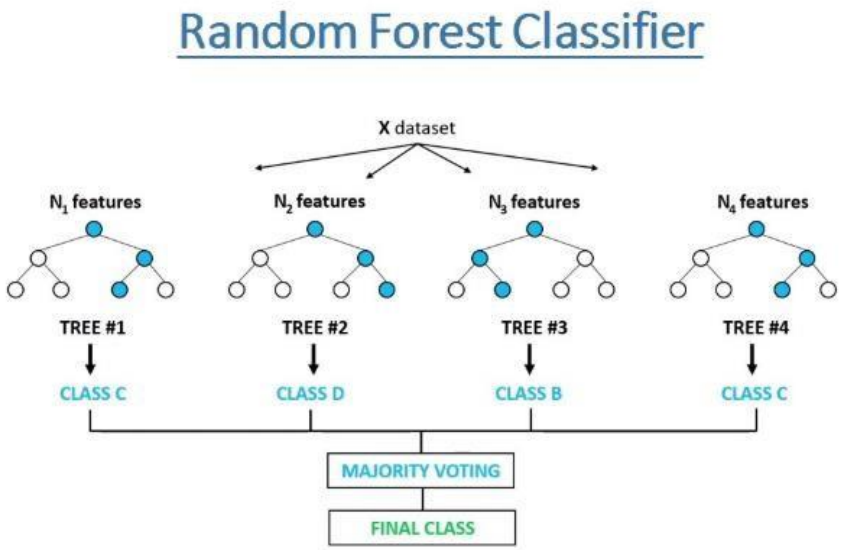

Fig. 5.3 – Random Forest Classifier

- **Support Vector Classifier (SVC):**

Support Vector Classifier (SVC) is a type of Support Vector Machine (SVM) used for classification tasks. SVMs are powerful supervised learning algorithms that can be used for both classification and regression challenges. In the context of disease diagnosis using machine learning, SVCs play a crucial role in accurately classifying patient data based on various medical features, ultimately aiding in early and precise disease detection.

The core idea behind SVC is to find the optimal hyperplane that best separates the data points of different classes in a high-dimensional space. This optimal hyperplane is chosen to maximize the margin between the classes, which is the distance between the nearest data points (support vectors) of each class and the hyperplane.

1. **Linear SVC**: In cases where the data is linearly separable, SVC finds a linear hyperplane that separates the classes.
2. **Non-Linear SVC**: For complex datasets where classes are not linearly separable, SVC uses kernel functions to transform the data into a higher-dimensional space where a linear hyperplane can be found.

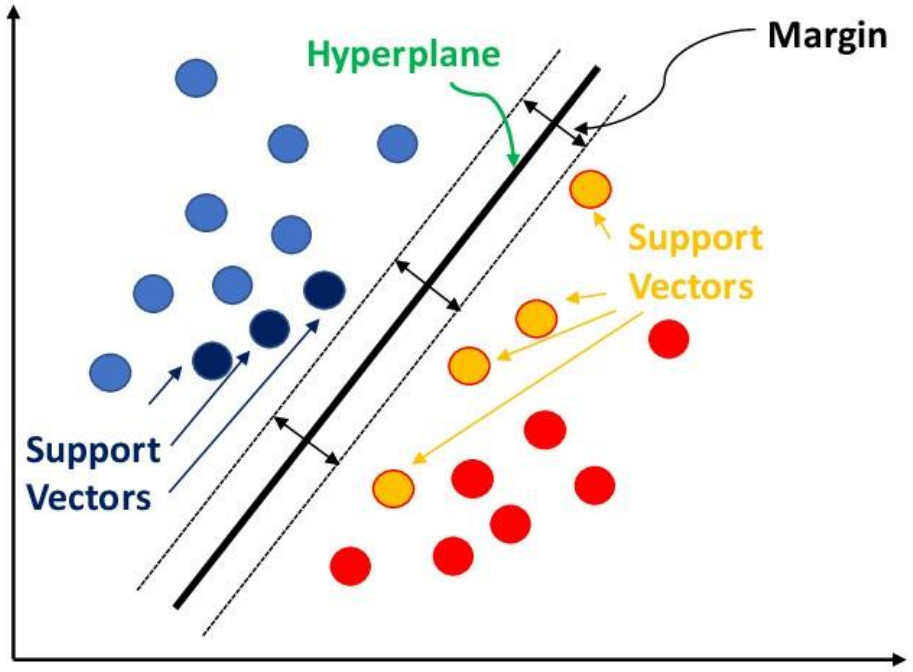

Fig. 5.4 – SVC

Implement each algorithm using Python libraries such as Scikit-learn, Seaborn, Matplotlib and Scipy.

5.1.5. Model Training :

- **Training Data**: Split the dataset into training and testing sets (e.g., 80% training, 20% testing).

- **Cross-Validation**: Use k-fold cross-validation (typically k=5 or k=10) to ensure the model's generalizability and to avoid overfitting. **5.1.6. Model Evaluation Evaluation Metrics:**

- **Accuracy**: Measure the proportion of correctly predicted instances.
- **Precision and Recall**: Evaluate the number of true positive results compared to the number of positive predictions made and the number of true positives compared to the actual number of positives.
- **F1 Score**: Calculate the harmonic mean of precision and recall to provide a single metric for model performance.
- **ROC-AUC**: Assess the area under the receiver operating characteristic curve to evaluate the trade-off between sensitivity and specificity.

Comparison:

- **Performance Analysis**: Compare the algorithms based on the evaluation metrics. Identify which model provides the best trade-off between bias and variance.
- **Statistical Tests**: Perform statistical tests (e.g., paired t-tests, Wilcoxon signed-rank tests) to determine if differences in performance are statistically significant.

5.1.7. Model Validation

- **Hold-out Validation**: Use a separate validation set that was not involved in training to test the model's performance.
- **External Validation**: Validate the model on external datasets to ensure robustness and generalizability.

5.2. Libraries Used:

5.2.1. NumPy:

NumPy is a powerful numerical computing library in Python that facilitates the efficient handling of arrays, matrices, and high-dimensional data structures. It forms the backbone of scientific computing and data analysis in Python due to its speed, versatility, and extensive capabilities.

1. **Array Operations:** NumPy's main data structure is the ndarray (n-dimensional array), which can be of any dimensionality. These arrays are homogeneous, meaning they contain elements of the same data type, allowing for efficient computation and memory management. NumPy provides a wide range of functions for creating, manipulating, and operating on arrays, including indexing, slicing, reshaping, and combining arrays.

2. **Vectorized Operations:** NumPy is designed for vectorized operations, where operations are applied element-wise to entire arrays without the need for explicit looping. This approach significantly speeds up computations, making NumPy ideal for handling large datasets and complex numerical calculations.

3. **Mathematical Functions:** NumPy offers a comprehensive suite of mathematical functions for numerical computations. This includes basic arithmetic operations like addition, subtraction, multiplication, and division, as well as advanced functions such as trigonometric functions, exponential and logarithmic functions, statistical functions, and linear algebra operations like matrix multiplication, inversion, and decomposition.

4. **Random Number Generation:** NumPy includes functions for generating random numbers and random samples from various probability distributions. This is useful for simulations, statistical analysis, and generating synthetic datasets.

5. **Integration with Scientific Libraries:** NumPy seamlessly integrates with other scientific computing libraries in Python, such as SciPy, pandas, matplotlib, and sci-kitlearn. This integration allows for a streamlined workflow for data manipulation, statistical analysis, visualization, and machine learning tasks.

6. **Performance:** NumPy's core operations are implemented in highly optimized C and Fortran code, making it blazingly fast compared to traditional Python code using lists. This performance advantage is particularly significant for numerical computations and data-intensive applications.

In essence, NumPy provides a robust foundation for numerical computing in Python, empowering users to perform complex mathematical operations, handle large datasets efficiently, and build sophisticated data analysis and machine learning pipelines.

5.2.2. Pandas:

Pandas is a powerful and popular library in Python for data manipulation and analysis. It provides data structures and functions to efficiently handle structured data, such as tabular data with rows and columns, making it ideal for tasks like data cleaning, transformation, exploration, and analysis. The core data structures in pandas are Series and DataFrame.

A Series is essentially a one-dimensional labelled array that can hold data of any type (integer, float, string, etc.). Each element in a Series has a corresponding label called an index, which can be customized or automatically generated. This makes it easy to access and manipulate data based on these labels.

On the other hand, a data frame is a two-dimensional labelled data structure resembling a spreadsheet or SQL table. It consists of rows and columns, where each column can be of a different data type. DataFrames can be created from various sources such as CSV files, Excel sheets, databases, or even manually from Python data structures like dictionaries or lists of lists.

Pandas offer a wide range of functionalities for data manipulation and analysis. Some key features include:

1. Data Cleaning: Pandas provides methods to handle missing data (NaN values), duplicate rows, and outliers. It also supports data type conversion, string manipulation, and data filtering.

2. Data Exploration: With pandas, you can perform descriptive statistics (mean, median, standard deviation, etc.), calculate correlations between variables, and visualize data using built-in plotting capabilities or integration with libraries like Matplotlib and Seaborn.

3. Data Transformation: Pandas allow for reshaping data using operations like pivoting, melting, and stacking/unstacking. You can also merge and concatenate DataFrames, perform group-by operations, and apply custom functions to data subsets.

4. Time Series Analysis: Pandas has robust support for working with time series data, including date/time indexing, resampling, frequency conversion, and time zone handling.

Overall, pandas simplifies the data analysis workflow in Python by providing intuitive and efficient tools for data manipulation, exploration, and transformation, making it a valuable library for data scientists, analysts, and researchers.

5.2.3. Scikit Learn:

Scikit-learn, often abbreviated as sklearn, is a comprehensive machine-learning library in Python that provides tools for data mining and data analysis. It is built on top of other popular scientific computing libraries like NumPy, SciPy, and matplotlib, making it a powerful and efficient choice for machine learning tasks.

1. **Model Selection**: The model_selection module in Scikit-learn includes utilities for model selection and evaluation. It provides tools for cross-validation, grid search, hyperparameter tuning, and model evaluation metrics like accuracy, precision, recall, F1score, ROC-AUC, and more. The train_test_split function is commonly used to split data into training and testing sets.

2. **Feature Selection**: Scikit-learn's feature_selection module offers methods for selecting relevant features and reducing dimensionality in datasets. It includes techniques like univariate feature selection, recursive feature elimination, feature importance estimation, and more, helping to improve model performance by focusing on the most informative features.

3. **Ensemble Methods**: The ensemble module implements ensemble learning techniques, where multiple models are combined to improve predictive performance. It includes

popular ensemble algorithms such as Random Forest, Gradient Boosting, AdaBoost, and Voting classifiers/regressors, offering flexibility and power in building robust predictive models.

4. **Preprocessing**: Scikit-learn's preprocessing module provides tools for data preprocessing and transformation. It includes functions for scaling features, encoding categorical variables, handling missing values, and more. StandardScaler, MinMaxScaler, OneHotEncoder, Imputer, and PolynomialFeatures are some commonly used preprocessing techniques.

5. **Metrics**: The metrics module in scikit-learn offers a wide range of evaluation metrics for assessing model performance. It includes metrics for classification tasks like accuracy, precision, recall, F1-score, ROC-AUC, and confusion matrix, as well as metrics for regression tasks like mean squared error, mean absolute error, R-squared score, and more.

Overall, sci-kit-learn's modular design, extensive documentation, and user-friendly interface make it a popular choice for both beginners and experienced practitioners in the field of machine learning, enabling efficient development, evaluation, and deployment of machine learning models.

5.2.4. Matplotlib:

Matplotlib is a comprehensive library in Python used for creating static, animated, and interactive visualizations. It provides a wide range of plotting functions and tools to generate high-quality plots for data analysis, scientific research, and data visualization tasks. Matplotlib's versatility and ease of use make it a popular choice among data scientists, researchers, engineers, and developers.

One of Matplotlib's key features is its support for various plot types, including line plots, scatter plots, bar plots, histograms, pie charts, 3D plots, and more. These plots can be customized extensively to suit specific requirements, such as adjusting colors, styles, labels, axes, legends, and annotations. Matplotlib also supports multiple subplots within a single figure, allowing users to create complex layouts and compare multiple datasets or visualizations side by side.

Matplotlib's architecture is designed to provide both a high-level interface for quick plotting tasks and a low-level interface for fine-grained control over plot elements. The high-level interface, often accessed through pyplot, allows users to create plots with minimal code and automatically handles many plot configurations. On the other hand, the low-level interface provides granular control over every aspect of the plot, making it suitable for advanced customization and specialized plotting requirements.

In addition to static plots, Matplotlib supports interactive plotting capabilities through backends like Qt, GTK, and Tkinter, enabling users to create interactive plots with zooming, panning, tooltips, and other interactive features. Furthermore, Matplotlib can be integrated seamlessly with other Python libraries, such as NumPy for data manipulation, pandas for data analysis, and SciPy for scientific computing, enhancing its capabilities and flexibility in data visualization tasks.

Overall, Matplotlib is a powerful and versatile library that empowers users to create publication-quality plots and visualizations, making it an essential tool for data exploration, presentation, and communication in the Python ecosystem.

5.2.5. Scipy:

SciPy is a powerful library in Python built on top of NumPy, focusing on scientific and technical computing. It provides a wide range of modules and functions for tasks such as optimization, integration, interpolation, signal processing, linear algebra, statistics, and more. Here's a breakdown of some key features and modules within SciPy:

1. **Optimization:** SciPy's optimize module offers various optimization algorithms for minimizing or maximizing objective functions, including methods like gradient descent, Newton's method, constrained optimization, and global optimization techniques.

2. **Integration:** The integrate module provides functions for numerical integration, including single and multiple integrals, as well as differential equation solvers like odeint for ordinary differential equations and ode for more general differential equations.

3. **Interpolation:** The interpolate module supports interpolation techniques such as spline interpolation, polynomial interpolation, and grid data interpolation, which are essential for curve fitting and data smoothing tasks.

4. **Signal Processing:** SciPy's signal module offers tools for digital signal processing, including filtering, Fourier transforms, wavelet transforms, convolution, correlation, and spectral analysis.

5. **Linear Algebra:** The linalg module provides functions for linear algebra operations, such as solving linear systems of equations, eigenvalue and eigenvector computations, matrix factorizations (LU, QR, SVD), and sparse matrix operations.

6. **Statistics:** SciPy's stats module includes a wide range of statistical functions and probability distributions, enabling tasks like hypothesis testing, probability density estimation, statistical modelling, and random variable generation.

7. **Special Functions:** The special module contains a collection of special mathematical functions, including Bessel functions, gamma functions, hypergeometric functions, and

orthogonal polynomials, which are commonly used in scientific and engineering calculations.

8. **Sparse Matrices:** SciPy provides efficient data structures and algorithms for working with sparse matrices through its sparse module, allowing for memory-efficient storage and manipulation of large sparse matrices commonly encountered in scientific computing.

Overall, SciPy complements NumPy by extending its capabilities to encompass a broader range of scientific computing tasks, making it an essential toolkit for researchers, engineers, data scientists, and anyone working on computational problems in Python.

5.2.6. Seaborn:

Seaborn is a powerful data visualization library built on top of Matplotlib in Python. It provides a high-level interface for creating attractive and informative statistical graphics. Seaborn is particularly useful for visualizing complex datasets and exploring relationships between variables.

One of Seaborn's key features is its ability to create visually appealing plots with minimal code. It offers a wide range of plot types, including scatter plots, line plots, bar plots, histograms, box plots, violin plots, heatmaps, and more. These plots are designed to effectively communicate insights from data, making them suitable for both exploratory data analysis and presentation purposes.

Seaborn also integrates seamlessly with pandas data structures, allowing for easy data manipulation and plotting. It provides functions like `sns.scatterplot()`, `sns.lineplot()`, `sns.barplot()`, `sns.histplot()`, `sns.boxplot()`, `sns.violinplot()`, `sns.heatmap()`, and many others, each tailored to specific visualization tasks. For example, `sns.scatterplot()` can be used to create scatter plots

with optional regression lines, while `sns.boxplot()` and `sns.violinplot()` are ideal for visualizing distributional information and comparing groups of data.

Moreover, Seaborn simplifies the process of creating complex multi-plot grids and conditional plots through its `FacetGrid` and `PairGrid` functionalities. These tools enable users to visualize relationships across multiple variables or subsets of data easily.

Another notable aspect of Seaborn is its support for statistical estimation and inference. It provides functions for visualizing statistical relationships with confidence intervals, performing categorical data analysis, and visualizing linear regression models with residuals.

Overall, Seaborn's combination of aesthetic appeal, ease of use, and statistical visualization capabilities makes it a popular choice for data scientists, analysts, and researchers looking to create informative and visually appealing plots to gain insights from their data.

CHAPTER 6

SYSTEM DESIGN

6.1. Flow chart of Model:

6.1.1. Download Dataset from Kaggle:

This step involves obtaining the dataset that will be used for developing the disease diagnosis model. Kaggle is a popular platform that hosts datasets for various machine learning problems. For this project, you would download a relevant medical dataset, such as one containing patient records with features that can help in diagnosing a particular disease.

6.1.2. Raw Data Input:

After downloading the dataset, the first step is to load the raw data into your environment. This typically involves reading the data file (e.g., CSV, Excel) into a pandas DataFrame. The raw data may contain missing values, duplicates, and other issues that need to be addressed in subsequent steps.

6.1.3. Preprocessing:

Data preprocessing is a crucial step in machine learning and data analysis pipelines. It involves transforming raw data into a format that is more suitable for analysis and modeling.

- **Null value removal:**
  In this step, you handle missing values in the dataset. This can be done by either removing rows or columns with null values or imputing them with appropriate values (mean, median, mode, or a value derived from other techniques).

- **Data Balancing:**

Often, medical datasets are imbalanced, meaning that the number of cases for different classes (e.g., diseased vs. healthy) is not equal. Data balancing techniques such as oversampling the minority class, under sampling the majority class, or using algorithms like SMOTE (Synthetic Minority Over-sampling Technique) can be used to address this issue.

6.1.4. Feature Engineering:

This involves creating new features or modifying existing ones to improve the model's performance. It can include transforming variables, creating interaction features, encoding categorical variables, and scaling numerical features.

6.1.5. Data Splitting:

Data splitting is a crucial step in machine learning where you divide your dataset into separate sets for training and testing your model. The purpose of data splitting is to evaluate the performance of your machine learning model on unseen data, which helps assess its ability to generalize to new, unseen instances.

- **Training set:**
  The training set is a subset of your dataset used to train the machine learning model. It contains input features (X) and corresponding target labels (y). The model learns patterns and relationships in the training data to make predictions.

  'x_train': Contains the input features for training.

  'y_train': Contains the corresponding target labels for training.

- **Testing set:**

The testing set, also known as the validation set or holdout set, is another subset of your dataset that remains unseen by the model during training. It is used to evaluate the model's performance and assess its ability to generalize to new data.

'x_test': Contains the input features for testing.

'y_test': Contains the corresponding target labels for testing.

6.1.6. Train dataset with Classification algorithm:

Train Dataset with Classification Algorithm: Train the model using a suitable classification algorithm. For disease diagnosis, common algorithms include Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines (SVM).

6.1.7. Evaluation:

Assess the model's performance using appropriate metrics such as accuracy, precision, recall, F1score, and ROC-AUC. Use a confusion matrix to understand the model's predictions.

6.1.8. Output:

Present the final results, which include the performance metrics and the trained model. Depending on the application, the model may be deployed for real-time predictions or further analyzed for insights.

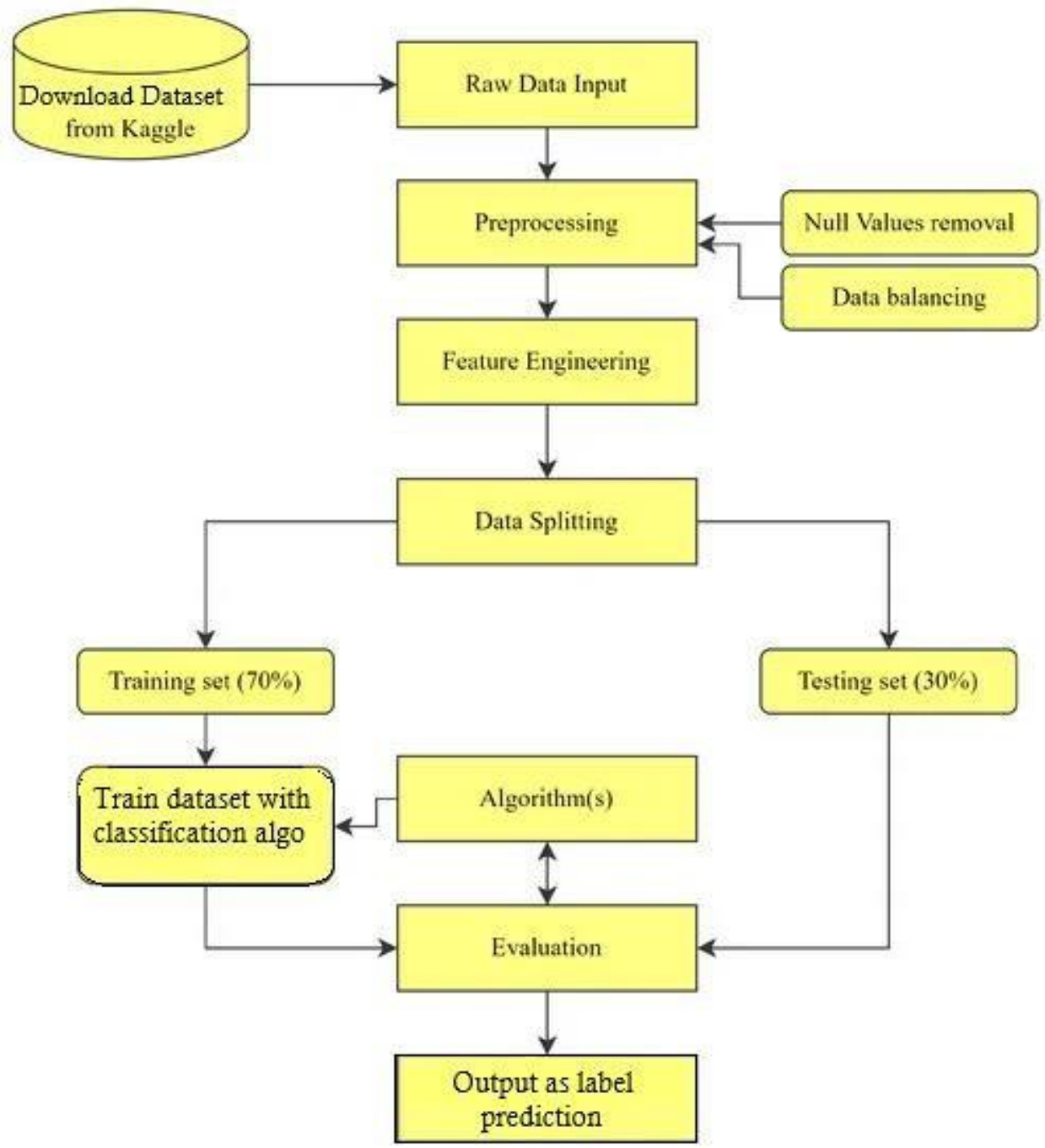

Fig. 5.4 – Flow Chart

CHAPTER 7

IMPLEMENTATION & VISUALIZATION

Table- 1:

Sr. N.	Disease Dataset	Number of Attributes	Number of Insatances	Type of Class Label
1	Breast Cancer	569	32	Binary
2	Lung Cancer	309	16	Binary
3	Heart Attack	1319	9	Binary
4	Diabetes	768	9	Binary

As in the source code, all major functions are created once for the breast cancer dataset and invoked for the other three datasets which are lung cancer, heart attack and diabetes. So, the section on Implementation and Visualization in this chapter is given for only one dataset that is breast cancer, as while other dataset functions and graphs are similar.

7.1 Implementation:

7.1.1 Importing various libraries and modules:

Various libraries and their modules are imported using the import keyword. Major libraries are numpy(used for work with arrays), pandas(used for data manipulation and analysis), sklearn(provide tools for data minig and analysis), matplotlib(used for graph ploting), scipy(is used for scientific and technical computations), etc.

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.feature_selection import chi2
from sklearn.ensemble import ExtraTreesClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import cross_val_score, StratifiedKFold
from scipy.stats import wilcoxon
```

7.1.2 Feature selection model for feature engineering:

A dataset may contain several features and not all features are necessary to predict the class label. To find those necessary features we need to perform feature selection, here ExtraTreeClassifier is used to select necessary features.

```
#appling the feature selection model to the dataset
etc = ExtraTreesClassifier()
etc.fit(breast_x, breast_y)

print(etc.feature_importances_)

feature_importance = pd.Series(etc.feature_importances_, index = breast_x.columns)
feature_importance.nlargest(12).plot(kind = 'barh')
plt.show()
```

7.1.3 Function to detect and remove outliers:

In a dataset, data is collected from various sources and there are chances for outliers. Outliers are very different from other values within a defined range of values. These outliers are required to be removed from the dataset. They can be treated by deleting or capping them using a normalization technique such as z_score, IQR, etc.

```
new_breast_feat = breast_feat.copy()
def outliers(df):
    # outlier detection and removal
    def detect_outliers_IQR(df, feature):
        Q1 = df[feature].quantile(0.25)
        Q3 = df[feature].quantile(0.75)
        IQR = Q3 - Q1
        lower_bound = Q1 - 1.5 * IQR
        upper_bound = Q3 + 1.5 * IQR
        return lower_bound, upper_bound

    # Define a function to cap outliers
    def cap_outliers(df, feature, lower_bound, upper_bound):
        df[feature] = np.where(df[feature] < lower_bound, lower_bound, df[feature])
        df[feature] = np.where(df[feature] > upper_bound, upper_bound, df[feature])
        return df

    # Loop through each feature in the DataFrame
    for feature in df.columns:
        # Detect outliers using IQR method
        lower_bound, upper_bound = detect_outliers_IQR(df, feature)

        # Cap outliers
        clean_df = cap_outliers(df, feature, lower_bound, upper_bound)

    # Print the resulting DataFrame
    return clean_df

breast_clean_df = outliers(new_breast_feat)
```

7.1.4 Train-Test Splitting and Data Standardization:

Train-test split is a technique to split the original dataset into training and testing datasets in the ratio of 75:25.

Data Standardization is used to standardize the data so that prediction model can be trained over the dataset with ease.

```
x_train, x_test, y_train, y_test = train_test_split(breast_clean_df, breast_y,
                                                    test_size = 0.25, random_state = 0)

sc = StandardScaler()

breast_x_train = sc.fit_transform(x_train)
breast_x_test = sc.fit_transform(x_test)
```

7.1.5 Train dataset with different prediction models:

Four Classification models are used for the prediction of class label, which are Logistic Regression, Decision Tree Classifier, Random Forest Classifier and Support vector Classifierr.

```python
def models(x_train, y_train):
    #Logistic regression
    from sklearn.linear_model import LogisticRegression
    breast_log = LogisticRegression(random_state = 0)
    breast_log.fit(breast_x_train, y_train)

    #Decision Tree
    from sklearn.tree import DecisionTreeClassifier
    breast_tree = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    breast_tree.fit(breast_x_train, y_train)

    #Random Forest Classifier
    from sklearn.ensemble import RandomForestClassifier
    breast_forest = RandomForestClassifier(n_estimators = 10, criterion ='entropy', random_state = 0)
    breast_forest.fit(breast_x_train, y_train)

    #Support Vector Classifier
    from sklearn.svm import SVC
    breast_svc = SVC(kernel = 'rbf', random_state = 0, probability = True)
    breast_svc.fit(breast_x_train, y_train)

    # print the model accuracy on the training data
    print('[0] Logistic Regression Training Accuracy: ', breast_log.score(breast_x_train , y_train))
    print('[1] Decision Tree Classifier Training Accuracy: ', breast_tree.score(breast_x_train , y_train))
    print('[2] Random Forest Classifier Training Accuracy: ', breast_forest.score(breast_x_train , y_train))
    print('[3] Support Vector Classifier Training Accuracy: ', breast_svc.score(breast_x_train , y_train))

    return breast_log, breast_tree, breast_forest, breast_svc

model = models(breast_x_train, y_train)
```

7.1.6 Function for cross-validation of models:

Cross-validation is used to cross-check the accuracy of the trained model over training dataset.

```python
# function for the cross-validation of models w.r.t their accuracy scoring
def cross_validation_evaluation(models, X, y, cv=5):
    skf = StratifiedKFold(n_splits=cv, shuffle=True, random_state=0)
    model_names = ['Logistic Regression', 'Decision Tree', 'Random Forest', 'SVC']

    for model, name in zip(models, model_names):
        scores = cross_val_score(model, X, y, cv=skf, scoring='accuracy')
        print(f'{name} Cross-Validation Accuracy: {scores.mean():.4f} (+/- {scores.std():.4f})')

# Evaluate models with cross-validation
cross_validation_evaluation(model, breast_clean_df, breast_y)
```

7.1.7 Test models accuracy on Test data with confusion matrix:

A confusion matrix a matrix that **summarizes the performance of a machine learning model on a set of test** data. It compares the actual target values against the ones predicted by the ML model.

Specificity also called the true negative rate, measures the proportion of actual negative cases that the model correctly identifies. It illustrates the model's ability to correctly identify the absence of the target variable.

```
#test model accuracy on test data on cofusion matrix
from sklearn. metrics import confusion_matrix

print('0 - Logistic Regression \n1 - Decision Tree Classifier \n2 - Random Forest Classifier \n3 - SVC')

for i in range(len(model)):
    print(model[i])
    cm = confusion_matrix(y_test, model[i].predict(breast_x_test))
    TP = cm[0][0]
    TN = cm[1][1]
    FP = cm[1][0]
    FN = cm[0][1]

    print(cm)
    print('Testing Accuracy = ', (TP+TN)/(TP+TN+FP+FN))
    print()

    if TN + FP == 0:
        print('0')
    else:
         print('Specificity - ',TN / (TN + FP))
```

7.1.8 Evaluation models by calculating Classification Report:

The classification report helps in understanding the performance of a classification model. It provides insights into how well the model is doing in terms of precision, recall, and F1-score.

```
#classification report
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score

for i in range(len(model)):
    print(model[i])
    print(classification_report(y_test, model[i].predict(breast_x_test)))
    print(accuracy_score(y_test, model[i].predict(breast_x_test)))
    print()
```

7.1.9 Using Wilcoxon Signed-Rank test for comparison of models by statistically testing the difference between them:

Wilcoxon Signed-Rank test is used to compare different prediction models by statistically testing them. The mean_difference is used to find the best-performing model out of two models.

```python
def wilcoxon_signed_rank_test(model1, model2, X_test, y_test):
    # Predict probabilities
    y_pred1 = model1.predict_proba(X_test)[:, 1]
    y_pred2 = model2.predict_proba(X_test)[:, 1]
    differences = y_pred1 - y_pred2

    # Perform Wilcoxon signed-rank test
    statistic, p_value = wilcoxon(differences)

    mean_difference = np.mean(differences)

    # Print results
    print(f'Wilcoxon Signed-Rank Test between models:\nStatistical Test:{statistic:.4f},P-value:{p_value:.4f}')
    print(f'Mean difference: {mean_difference:.4f}')

    if p_value < 0.05:
        if mean_difference > 0:
            print("Model 1 is performing better than Model 2.")
            print()
        else:
            print("Model 2 is performing better than Model 1.")
            print()
    else:
        print("There is no statistically significant difference between the two models.")
        print()

    return statistic, p_value, mean_difference
```

7.2 Visualization:

7.2.1 Heatmap- used to find the correlation between features:

This heatmap shows the correlation matrix for various wine attributes. Each cell represents the correlation coefficient between two variables, with color intensity indicating the strength and direction of the correlation. Darker shades signify stronger correlations, whether positive or negative.

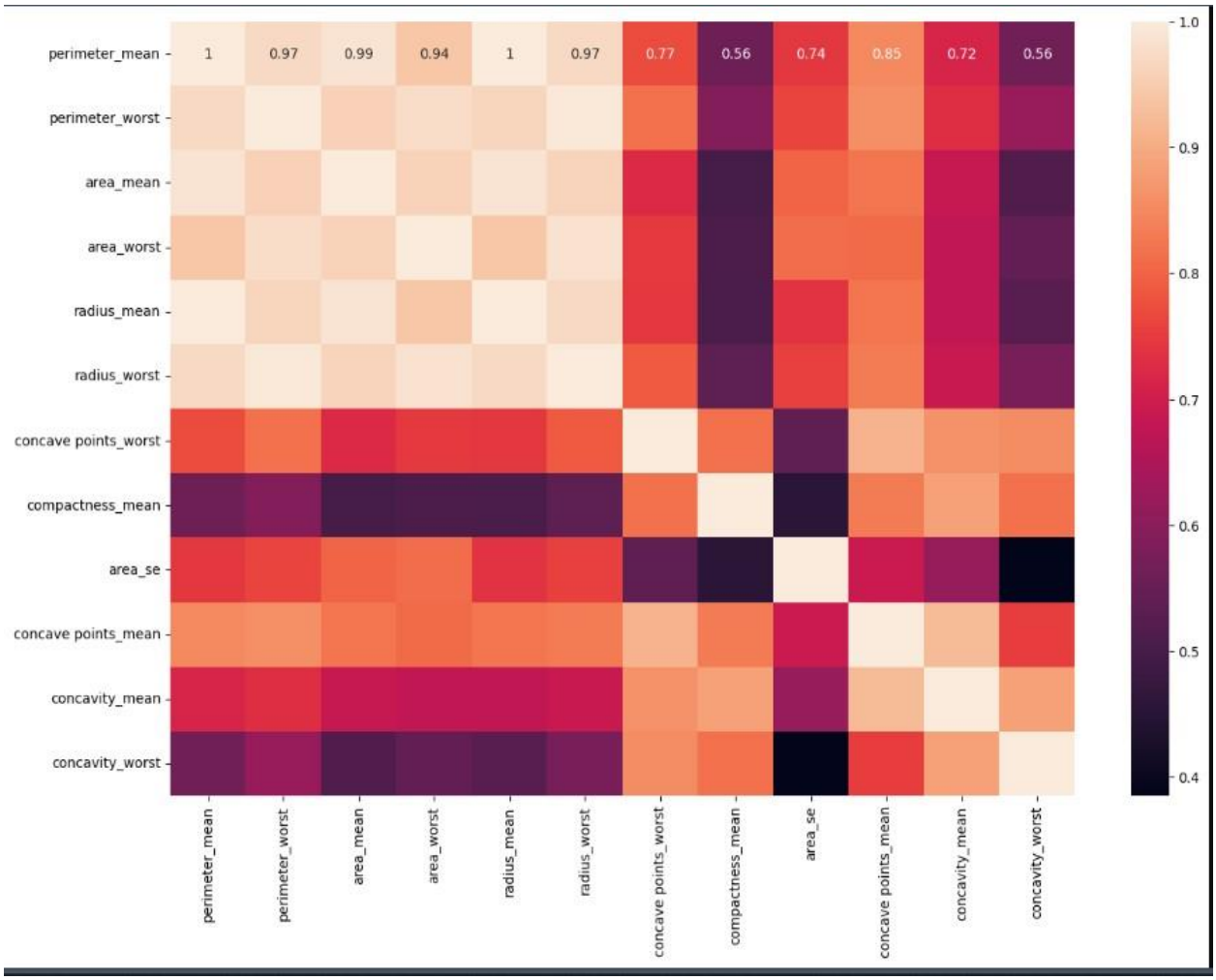

Fig. 7.1 – Heatmap

7.2.2 Boxplot – used to find the existence of outliers in the data:

A boxplot, also known as a box-and-whisker plot, is a standardized way of displaying the distribution of data based on a five-number summary: minimum, first quartile (Q1), median (Q2), third quartile (Q3), and maximum. Boxplot is used for identifying outliers.

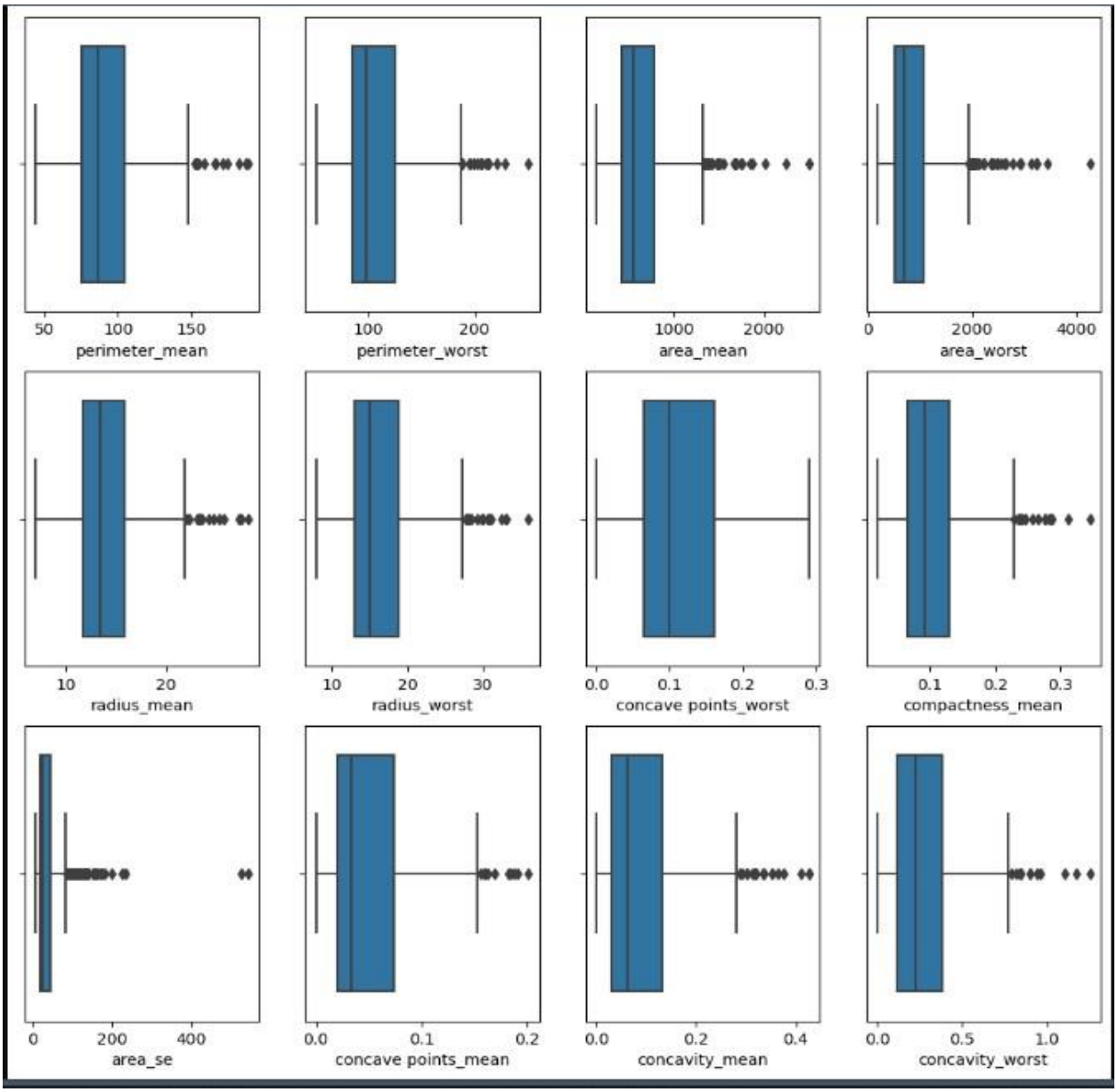

Fig. 7.2 – Boxplot

7.2.3 Histplot – to find the skewness of data:

The histplot function in the Seaborn library is used to plot histograms. It is a versatile tool for visualizing the distribution of a single variable or comparing the distributions of multiple

variables. Histograms show the frequency of data points within specific ranges, providing insights into the distribution, central tendency, and spread of the data.

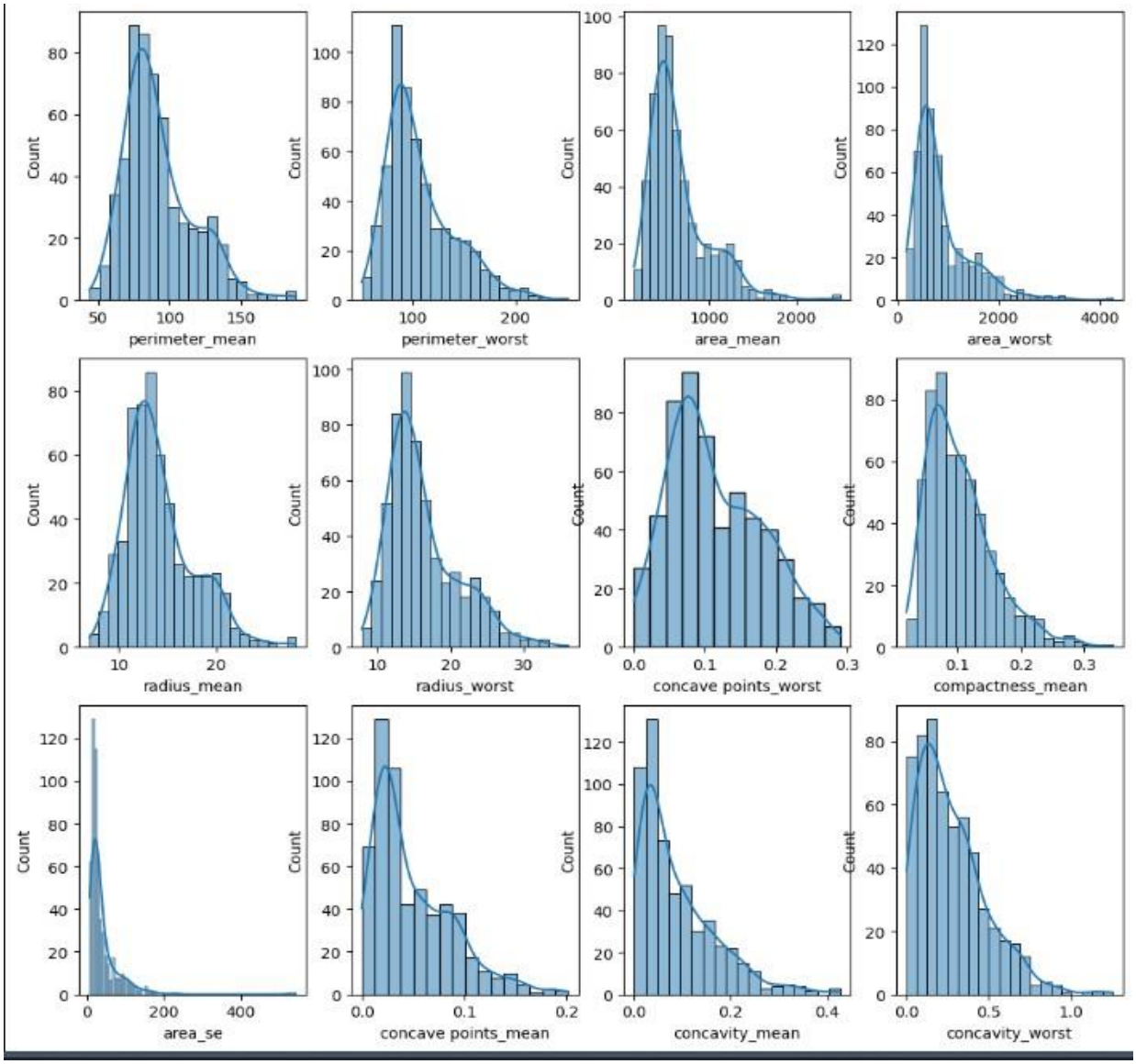

Fig. 7.3 – Histplot

CHAPTER 8

DISCUSSION AND ANALYSIS OF RESULT

The code provided outlines a comprehensive workflow for breast cancer prediction using various machine learning techniques. Here, I will break down the result and analysis of each major step in the process.

1. Data Loading and Initial Exploration

The dataset is loaded from a CSV file, and initial exploration includes displaying the first few rows, shape, info, summary statistics, and checking for null values. This ensures the dataset is properly loaded and ready for preprocessing.

2. Data Preprocessing

- **Null Value Removal:** No null values were found in the dataset.
- **Label Encoding:** The target variable 'diagnosis' is encoded from categorical to numerical.

3. Feature Engineering

- **Feature Importance:** An ExtraTreesClassifier is used to identify the most important features.
    - The top 12 features are visualized using a bar plot, and their correlation matrix is plotted using a heatmap.
- **Outlier Detection and Removal:** Outliers in the top 12 features are capped using the IQR method to improve model performance.

4. Data Splitting

- The cleaned dataset is split into training (75%) and testing (25%) sets.

5. Data Scaling

- Feature scaling is applied to ensure all features contribute equally to the model training.

6. Model Training

Four different models are trained using the training data:

1. Logistic Regression
2. Decision Tree Classifier
3. Random Forest Classifier
4. Support Vector Classifier (SVC)

7. Model Evaluation

- **Training Accuracy:** The training accuracy of each model is printed.
- **Cross-Validation:** The models are evaluated using 5-fold cross-validation to ensure robustness.
- **Testing Accuracy:** Confusion matrices and testing accuracy are computed for each model.
- **ROC and AUC:** The ROC curves and AUC scores are plotted to visualize and compare the performance of the models.

8. Model Comparison

- **Wilcoxon Signed-Rank Test:** This non-parametric test is used to compare the performance of different models on the test set. The test evaluates if there is a statistically significant difference between pairs of models.

Results

1. Result Table:
   1.1 Breast Cancer:

	models	Accuracy	Cross-validation	Specificity	Recall(0)	Recall(1)	f1-score(0)	f1-score(1)	AUC
0	Logistic Regression	0.944055	94.55% (+/- 0.02)	0.962264	0.93	0.96	0.95	0.93	0.994
1	Decision Tree Classifier	0.930069	92.27% (+/- 0.02)	0.905660	0.94	0.91	0.94	0.91	0.925
2	Random Forest Classifier	0.951048	94.35% (+/- 0.03)	0.943396	0.96	0.94	0.96	0.93	0.994
3	Support Vector Classifier	0.944055	91.75% (+/- 0.03)	0.943396	0.94	0.94	0.96	0.93	0.991

1.2 Lung Cancer:

	models	Accuracy	Cross-validation	Specificity	Recall(0)	Recall(1)	f1-score(0)	f1-score(1)	AUC
0	Logistic Regression	0.923076	91.59% (+/- 0.01)	0.970588	0.6	0.97	0.67	0.96	0.956
1	Decision Tree Classifier	0.858974	88.67% (+/- 0.04)	0.955882	0.2	0.96	0.27	0.92	0.578
2	Random Forest Classifier	0.897435	89.64% (+/- 0.02)	0.955882	0.5	0.96	0.56	0.94	0.933
3	Support Vector Classifier	0.884615	87.38% (+/- 0.02)	0.970588	0.3	0.97	0.40	0.94	0.951

1.3 Heart Attack:

	models	Accuracy	Cross-validation	Specificity	Recall(0)	Recall(1)	f1-score(0)	f1-score(1)	AUC
0	Logistic Regression	0.924242	81.73% (+/- 0.01)	0.943298	0.90	0.94	0.91	0.94	0.981
1	Decision Tree Classifier	0.912121	97.95% (+/- 0.00)	0.979381	0.82	0.98	0.88	0.93	0.976
2	Random Forest Classifier	0.918181	98.56% (+/- 0.00)	0.989690	0.82	0.99	0.89	0.93	0.898
3	Support Vector Classifier	0.906060	63.00% (+/- 0.01)	0.860824	0.97	0.86	0.89	0.92	0.922

1.4 Diabetes:

	models	Accuracy	Cross-validation	Specificity	Recall(0)	Recall(1)	f1-score(0)	f1-score(1)	AUC
0	Logistic Regression	0.796875	76.44% (+/- 0.02)	0.884615	0.61	0.88	0.66	0.85	0.861
1	Decision Tree Classifier	0.734375	68.61% (+/- 0.03)	0.784615	0.63	0.78	0.60	0.80	0.840
2	Random Forest Classifier	0.755208	73.03% (+/- 0.03)	0.769230	0.73	0.77	0.66	0.81	0.707
3	Support Vector Classifier	0.786458	75.92% (+/- 0.02)	0.876923	0.60	0.88	0.64	0.85	0.800

2. **Wilcoxon Signed-Rank Test:**

2.1 Breast Cancer: ○ Logistic Regression vs Decision Tree: Logistic Regression performed significantly better.

- Logistic Regression vs Random Forest: Logistic Regression performed significantly better.
- Logistic Regression vs SVC: No significant difference. ○ Decision Tree vs Random Forest: Random Forest performed significantly better. ○ Decision Tree vs SVC: SVC performed significantly better. ○ Random Forest vs SVC: No significant difference.

2.2 Lung Cancer:

- Logistic Regression vs Decision Tree: Decision Tree performed significantly better.
- Logistic Regression vs Random Forest: No significant difference. ○ Logistic Regression vs SVC: Logistic Regression performed significantly better. ○ Decision Tree vs Random Forest: Decision Tree performed significantly better. ○ Decision Tree vs SVC: Decision Tree performed significantly better. ○ Random Forest vs SVC: No significant difference.

2.3 Heart Disease:

- Logistic Regression vs Decision Tree: No significant difference. ○ Logistic Regression vs Random Forest: No significant difference.
- Logistic Regression vs SVC: No significant difference.
- Decision Tree vs Random Forest: No significant difference. ○ Decision Tree vs SVC: No significant difference. ○ Random Forest vs SVC: No significant difference.

2.4 Diabetes:

- Logistic Regression vs Decision Tree: No significant difference. ○ Logistic Regression vs Random Forest: No significant difference. ○ Logistic Regression vs SVC: No significant difference. ○ Decision Tree vs Random Forest: No significant difference. ○ Decision Tree vs SVC: No significant difference. ○ Random Forest vs SVC: No significant difference.

Analysis:

Upon reviewing the testing accuracy, precision, f1-score, specificity, recall, and AUC, it has been determined that Logistic Regression outperforms the other four models in predicting class labels.

CHAPTER 9

CONCLUSION

The development and application of machine learning models for disease prediction, such as the breast cancer model detailed above, offer significant future opportunities. Comparative analysis across datasets for breast cancer, lung cancer, heart attack, and diabetes can optimize models and enhance predictive capabilities.

Conclusion:

The future of disease prediction models is promising, driven by machine learning, data integration, and personalized medicine. By leveraging comparative analysis across various disease datasets, we can develop accurate, robust, and generalizable models. These advancements will enhance early diagnosis, treatment, and pave the way for innovative healthcare solutions, significantly improving patient outcomes and quality of life.

REFERENCES

Links:

1. (PDF) Performance Evaluation and Comparative Analysis of Different Machine Learning Algorithms in Predicting Cardiovascular Disease (researchgate.net)
2. Machine Learning Algorithms (geeksforgeeks.org)
3. www.youtube.com
4. https://www.researchgate.net
5. Find Open Datasets and Machine Learning Projects | Kaggle

6. Gaurav Singh, "Breast Cancer Prediction Using Machine Learning", International Journal of Scientific Research in Computer Science, Engineering and Information Technology (IJSRCSEIT), ISSN: 2456-3307, Volume 6 Issue 4, pp. 278-284, July August 2020.
   Available at doi: https://doi.org/10.32628/CSEIT206457
   Journal URL: http://ijsrcseit.com/CSEIT206457

7. Ahsan MM, Luna SA, Siddique Z. Machine-Learning-Based Disease Diagnosis: A Comprehensive Review. Healthcare (Basel). 2022 Mar 15;10(3):541. doi: 10.3390/healthcare10030541. PMID: 35327018; PMCID: PMC8950225.

8. Harshit Jindal[1], Sarthak Agrawal[1], Rishabh Khera[1], Rachna Jain[2] and Preeti Nagrath[2]
   Published under licence by IOP Publishing LtdIOP Conference Series: Materials Science and Engineering, Volume 1022, 1st International Conference on Computational Research and Data Analytics (ICCRDA 2020) 24th October 2020, Rajpura, India**Citation** Harshit Jindal *et al* 2021 *IOP Conf. Ser.: Mater. Sci. Eng.* **1022** 012072**DOI** 10.1088/1757899X/1022/1/012072

9. A. S. Afridi, M. Abdullah-Al-Kafi, W. Sabbir, M. S. Rahman, N. P. Stenin and D. M.

Raza, "Comparative Analysis of Machine Learning Algorithms for Predicting Heart Disease: A Comprehensive Study," 2024 11th International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2024, pp. 1265-1270, doi: 10.23919/INDIACom61295.2024.10498611.

10. Uddin, S., Khan, A., Hossain, M. E., & Moni, M. A. (2019). Comparing different supervised machine learning algorithms for disease prediction. *BMC Medical Informatics and Decision Making*, *19*. https://doi.org/10.1186/s12911-019-1004-8