**Heart disease prediction using hybrid learning for multimodal ECG dataset**

**SYNOPSIS**

Heart disease remains a leading cause of mortality worldwide, emphasizing the importance of accurate and timely diagnosis. This project proposes a novel approach for heart disease prediction by leveraging hybrid learning techniques applied to multimodal electrocardiogram (ECG) datasets. By integrating information from multiple ECG modalities, we aim to enhance the accuracy and robustness of predictive models, enabling more effective early detection and intervention. The primary objective of our research is to develop a comprehensive framework for heart disease prediction that leverages the complementary strengths of various machine learning (ML) algorithms and modalities of ECG data.

**SYSTEM ENVIRONMENT**

2.1 Hardware Requirements:

Processor : Intel Core i4 (10th Gen)

Ram : 4.0 GB

2.2 Software Requirements

Operating System : Windows 10

Framework : Google colab

Language : python

**2.3 About the technology:**

Python:

Python is an interpreted high-level general-purpose programming language created by Guido Van Rossum and first published in 1991. Python's design philosophy emphasizes code readability with significant whitespace. Its language structures and object-oriented approach are designed to help developers write clear and logical code for small and large projects. Python is dynamically typed and garbage

Google Colab:

Google Colab, short for Google Colaboratory, is a cloud-based, interactive computing platform provided by Google. It allows users to write and execute Python code in a collaborative and convenient environment directly through a web browser. Colab provides free access to GPU and TPU (Tensor Processing Unit) resources, enabling accelerated execution of machine learning tasks. Users can create and share Jupyter notebooks, incorporating text, code, and visualizations seamlessly. Colab integrates with Google Drive, facilitating easy storage and sharing of notebooks. Its collaborative features enable multiple users to work on the same document simultaneously, fostering collaborative research and development. Overall, Google Colab is a powerful and accessible tool for data analysis, machine learning, and collaborative coding, making it particularly valuable for researchers, students, and practitioners in the field of data science.

Scikit Learn:

Scikit-learn (Sklearn) is the most useful and powerful Python machine learning library. It provides a number of powerful tools for machine learning and statistical modelling, including classification, regression, clustering and dimensionality reduction through a Python consistent interface. Written mostly in Python, this library is built on top of NumPy, SciPy and Matplotlib. Originally called scikits. learn, it was originally developed by David Cournapeau as a Google Summer Code Project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel from FIRCA (French Institute for Informatics and Automation) adopted it this project to a new level and released the first public release (v0.1 beta) on February 1, 2010

**EXISTING SYSTEM**

The existing system for Heart disease prediction using hybrid learning for multimodal ECG datasettypically follows a structured process:

Multimodal ECG signals are collected from patients using wearable devices or standard ECG machines. Preprocessing techniques are applied to clean the data, remove noise, and normalize signals to ensure consistency across different recordings. Features are extracted from the ECG signals to capture relevant information about cardiac activity. This includes traditional features such as QRS complex duration, ST segment elevation/depression, T wave morphology, as well as more advanced features derived from heart rate variability analysis, frequency domain analysis, and time-frequency representations. Labeled datasets, such as the PTB Diagnostic ECG Database or other publicly available datasets, are used for training and evaluation. The dataset is divided into training, validation, and test sets to assess model performance accurately. Machine learning algorithms, such as support vector machines (SVM) are trained on the extracted features. Models are optimized using techniques like cross-validation, hyperparameter tuning, and regularization to improve performance and prevent overfitting.

The trained models are evaluated using performance metrics such as accuracy, sensitivity, specificity, and area under the receiver operating characteristic curve (AUC-ROC). Performance is assessed on the validation and test sets to ensure generalization to unseen data. Once validated, the models can be deployed in clinical settings to assist healthcare professionals in diagnosing cardiovascular diseases. Integration with electronic health record systems or mobile health applications enables real-time monitoring and decision support.

The system undergoes continuous monitoring and refinement based on feedback from clinicians and updates to the underlying data and algorithms.

New research findings and advances in machine learning techniques are incorporated to enhance diagnostic accuracy and clinical utility.

This system offers a non-invasive, scalable approach to cardiovascular disease diagnosis, leveraging machine learning and multimodal ECG signals to assist healthcare providers in early detection and personalized management of cardiac conditions.

**PROPOSED SYSTEM**

Here's a proposed system for heart disease prediction using hybrid learning for a multimodal ECG dataset, incorporating the MIT-BIH Arrhythmia Database (MITBIH), the PTB Diagnostic ECG Database (PTBDB), and training with decision tree, random forest, and ensemble classifier algorithms:

Concatenate the MIT-BIH Arrhythmia Database (MITBIH) and the PTB Diagnostic ECG Database (PTBDB) normal and abnormal files to create a comprehensive dataset. Preprocess the data, which may involve normalization, denoising, and feature extraction from the ECG signals. For multimodal data, extract features from each modality (e.g., ECG waveform, heart rate variability) separately. Develop a hybrid learning model that combines decision tree, random forest, and ensemble classifier algorithms.

For decision tree and random forest models, set hyperparameters such as tree depth, number of trees, and splitting criteria. For ensemble classifiers, consider methods like AdaBoost or Gradient Boosting to combine the predictions of multiple base learners. Train the hybrid model using the concatenated dataset, jointly considering all four files (MITBIH train and test, PTBDB normal and abnormal). Split the dataset into training and testing sets using techniques like cross-validation to ensure robust evaluation.

Evaluate the performance of each algorithm using metrics such as accuracy, precision, recall, F1-score. Generate confusion matrices to visualize the model's performance in predicting true positive, true negative, false positive, and false negative cases. Calculate classification vectors to assess the distribution of predicted classes and compare them across algorithms.

Analyze the results to identify the most effective algorithm for heart disease prediction based on the multimodal ECG dataset. Interpret the findings, considering factors such as model accuracy, interpretability, and computational efficiency.

By following this proposed system, you can leverage hybrid learning techniques to effectively predict heart disease using multimodal ECG data, while also comparing the performance of decision tree, random forest, and ensemble classifier algorithms for this task.

**Advantages of the proposed system:**

The proposed system for heart disease prediction using hybrid learning for multimodal ECG dataset offers several advantages.

**Comprehensive Data Integration:**

By concatenating data from multiple sources (MITBIH and PTBDB), the system incorporates a wide range of ECG signals, including normal and abnormal cases, leading to a more comprehensive and representative dataset.

**Enhanced Feature Representation:**

Multimodal data processing allows for the extraction of diverse features from different aspects of ECG signals, such as waveform morphology, heart rate variability, and other relevant parameters. This comprehensive feature representation can improve the discriminative power of the models.

**Synergistic Model Fusion:**

Hybrid learning combines multiple machine learning algorithms (decision tree, random forest, ensemble classifiers) to leverage their individual strengths and compensate for weaknesses. This synergistic fusion enhances predictive performance and robustness.

**Model Diversity and Stability:**

Ensemble classifiers, such as AdaBoost or Gradient Boosting, promote model diversity by sequentially training weak learners on the residuals of previous iterations. This enhances model stability and reduces overfitting, leading to more reliable predictions.

**Interpretability and Explainability:**

Decision tree models offer interpretability by generating intuitive decision rules that can be easily understood by domain experts. This transparency facilitates model interpretation and enables insights into the factors influencing heart disease prediction.

**Versatility and Flexibility:**

The proposed system can be adapted and extended to accommodate additional datasets or modalities, allowing for scalability and versatility in addressing diverse research questions or clinical applications related to heart disease prediction.

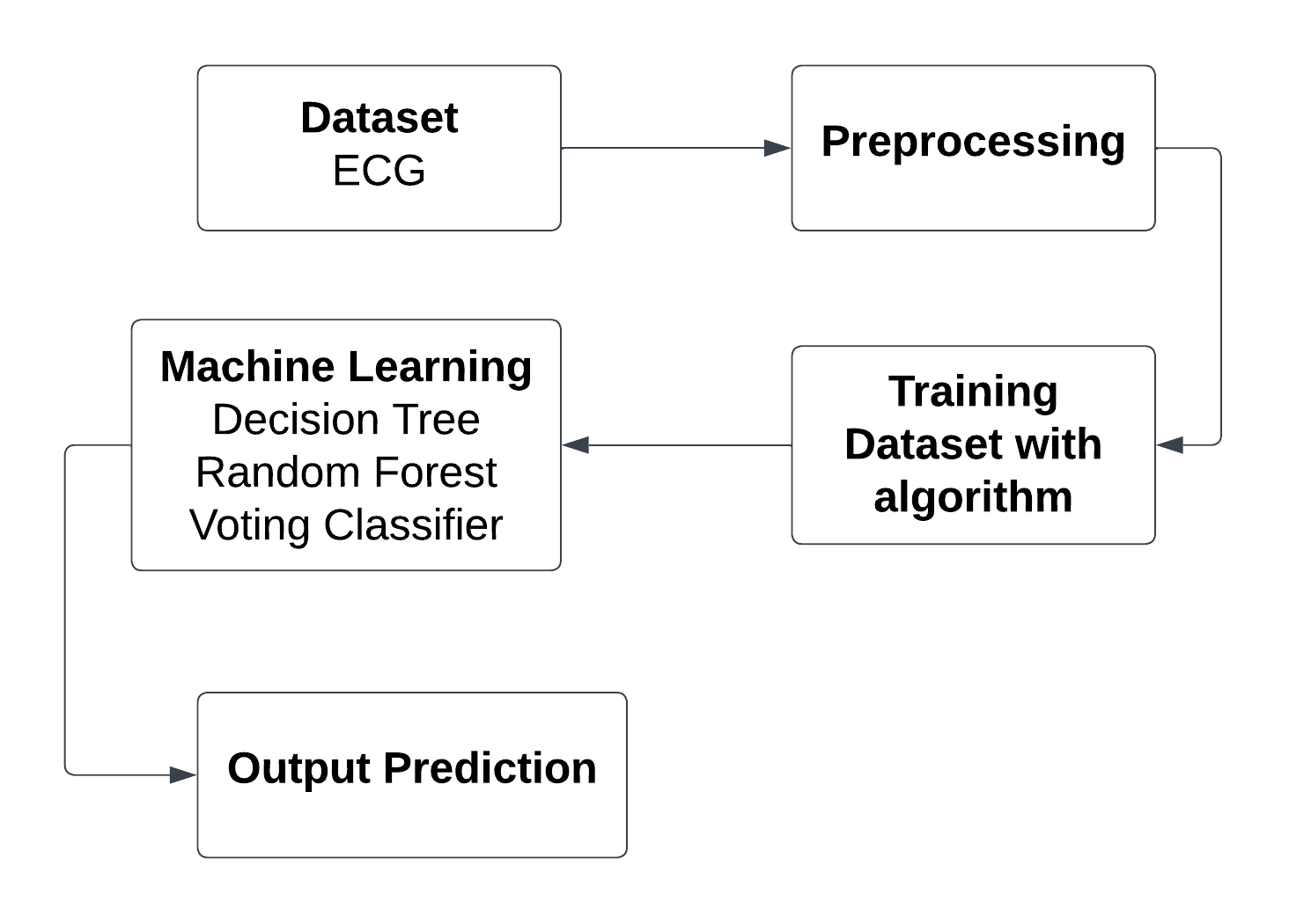
**Performance Evaluation:**

The use of metrics such as confusion matrices, classification vectors, accuracy, precision, recall, F1-score, and AUC enables comprehensive performance evaluation, providing insights into the strengths and limitations of each algorithm and guiding further refinement and optimization efforts.

The proposed system offers a holistic approach to heart disease prediction, leveraging the advantages of multimodal data integration and hybrid learning to enhance predictive performance, interpretability, and clinical relevance.

**SYSTEM DESIGN**

Heart Disease Prediction using hybrid learning for multimodal ECG dataset is designed by the below systematic diagram:



**Dataset Description:**

The dataset you mentioned consists of two main components: the MIT-BIH Arrhythmia Database (MITBIH) and the PTB Diagnostic ECG Database (PTBDB), each containing normal and abnormal ECG recordings. Here's a description of each component:

**MIT-BIH Arrhythmia Database (MITBIH):**

The MIT-BIH Arrhythmia Database is a widely used dataset containing ECG recordings collected from a variety of sources, including normal subjects and patients with various arrhythmias. It includes two main subsets: the training set (MITBIH Train) and the test set (MITBIH Test).

Each recording is typically represented as a time-series signal, sampled at a frequency of 360 Hz, with annotations indicating the presence of different types of arrhythmias, such as atrial fibrillation, ventricular tachycardia, and others.

Annotations are provided by expert cardiologists and serve as ground truth labels for training and evaluation of machine learning algorithms.

**PTB Diagnostic ECG Database (PTBDB):**

The PTB Diagnostic ECG Database is another dataset containing ECG recordings collected from healthy individuals (normal) and patients with various cardiac conditions (abnormal), compiled by the Physikalisch-Technische Bundesanstalt (PTB) in Germany.It consists of two classes: normal and abnormal, with each class represented by multiple recordings.

ECG recordings in the PTBDB are annotated with diagnostic labels, providing information about the presence of specific cardiac conditions or abnormalities, such as myocardial infarction, ischemia, and others.

The combined dataset comprises a diverse collection of ECG recordings from both healthy individuals and patients with various cardiac conditions, facilitating research and development in areas such as arrhythmia detection, heart disease diagnosis, and cardiac risk stratification. Researchers and clinicians can leverage this dataset to develop and evaluate machine learning algorithms for automated ECG analysis, aiding in the early detection and management of cardiac disorders.

**Pre-Processing:**

In the preprocessing stage, the four files containing ECG data (MITBIH train, MITBIH test, PTBDB normal, and PTBDB abnormal) are concatenated into a single dataset. Each ECG recording is standardized to a common format and length, ensuring consistency across the dataset. Additionally, any necessary preprocessing steps such as noise removal, baseline correction, and resampling may be applied to enhance the quality of the ECG signals. Finally, feature extraction techniques are employed to capture relevant information from the ECG data, preparing it for training machine learning models for heart disease prediction.

**Machine learning algorithm**

**1.Random Forest**

Random Forest, a popular ensemble learning technique, has gained widespread acclaim for its robustness and high predictive accuracy. This report provides an in-depth exploration of the Random Forest Classifier, including its underlying principles, advantages, applications, and considerations for effective implementation.

Principles:

Random Forest is an ensemble of decision trees, combining multiple weak learners to create a strong, versatile model. Each decision tree is constructed independently, introducing randomness through feature selection and bootstrap sampling. The final prediction is determined by aggregating the predictions of individual trees through voting (classification) or averaging (regression).

Advantages:

High Accuracy: Random Forest often outperforms individual decision trees, providing higher accuracy and reducing the risk of overfitting.

Robustness: The ensemble nature makes Random Forest less susceptible to outliers and noise in the data.

Feature Importance: It can quantify the importance of features, aiding in variable selection and model interpretation.

Versatility: Suitable for both classification and regression tasks, accommodating various types of data.

Applications:

Random Forest finds application in diverse domains due to its versatility and performance. Some notable applications include:

Finance: Credit scoring, fraud detection.

Healthcare: Disease prediction, patient outcome analysis.

Marketing: Customer churn prediction, targeted advertising.

Remote Sensing: Land cover classification, object detection.

Manufacturing: Quality control, predictive maintenance.

Considerations:

Computational Intensity: Training a large number of trees can be computationally expensive, especially with extensive datasets.

Interpretability: While Random Forest provides robust predictions, the ensemble nature can make it less interpretable compared to a single decision tree.

Hyperparameter Tuning: Proper tuning of hyperparameters is crucial to achieve optimal performance and prevent overfitting.

Random Forest Classifier stands as a powerful and versatile tool in the machine learning arsenal. Its ability to handle complex relationships in data, high accuracy, and resilience to overfitting make it a go-to choose for many practitioners. Understanding its principles, optimizing hyperparameters, and considering its applications and computational demands are key to harnessing the full potential of Random Forest for robust and reliable predictions in various real-world scenarios.

**2. Decision Tree Classifier**

Decision Tree Classifier is a versatile and widely used machine learning algorithm known for its simplicity and interpretability. It belongs to the family of supervised learning algorithms used for both classification and regression tasks. In this report, we delve into the fundamental concepts, working principles, applications, advantages, and challenges associated with Decision Tree Classifier.

**Working Principles:**

At its core, a Decision Tree is a flowchart-like structure where each node represents a feature or attribute, each branch represents a decision rule, and each leaf node represents an outcome or a class label. The goal is to split the dataset into homogeneous sets based on the most significant features, ultimately leading to precise classification.

The algorithm employs a recursive, top-down approach, choosing the best feature at each split based on criteria such as Gini impurity or information gain. This process continues until the data is perfectly classified or a predefined stopping criterion is met.

**Applications:**

Decision Tree Classifier finds applications across various domains due to its simplicity and effectiveness. Some notable applications include:

Finance: Predicting creditworthiness and fraud detection.

Medicine: Identifying diseases based on patient data.

Marketing: Customer segmentation and targeted advertising.

Manufacturing: Quality control and fault detection.

Agriculture: Crop disease prediction and yield estimation.

**Advantages:**

Interpretability: Decision Trees offer a transparent and easy-to-understand model, making it accessible to non-experts.

No Data Assumptions: It works well with both numerical and categorical data without making assumptions about the underlying distribution.

Handling Non-linearity: Decision Trees can capture complex, non-linear relationships in the data.

Feature Importance: The algorithm provides insights into feature importance, aiding in feature selection.

**Challenges:**

Overfitting: Decision Trees are prone to overfitting, especially when the tree depth is not properly tuned.

Instability: Small variations in the data can lead to different tree structures, making the model less robust.

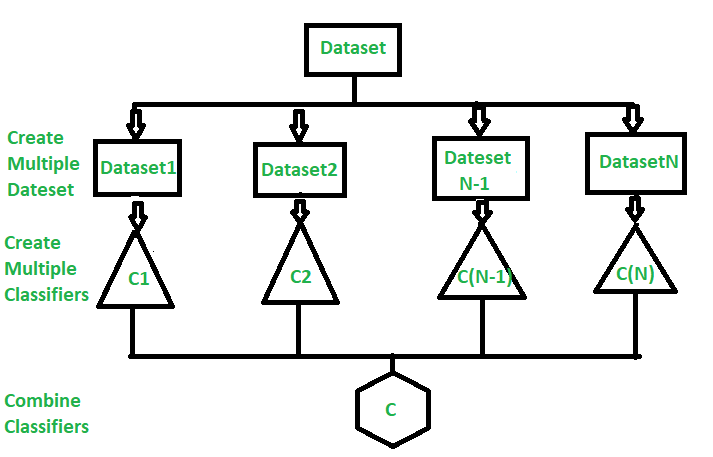
Bias Towards Dominant Classes: In imbalanced datasets, Decision Trees may favor the majority class.

Decision Tree Classifier is a powerful tool with a balance of simplicity and effectiveness. Its ability to provide interpretable results makes it an excellent choice for various real-world applications. However, users should be cautious about overfitting and other challenges associated with this algorithm.

**3. Ensemble Classifier**

Ensemble learning is a machine learning technique that enhances accuracy and resilience in forecasting by merging predictions from multiple models. It aims to mitigate errors or biases that may exist in individual models by leveraging the collective intelligence of the ensemble.

The underlying concept behind ensemble learning is to combine the outputs of diverse models to create a more precise prediction. By considering multiple perspectives and utilizing the strengths of different models, ensemble learning improves the overall performance of the learning system. This approach not only enhances accuracy but also provides resilience against uncertainties in the data. By effectively merging predictions from multiple models, ensemble learning has proven to be a powerful tool in various domains, offering more robust and reliable forecasts.



In this section, we will look at a few simple but powerful techniques, namely:

Max Voting

Averaging

Weighted Averaging

**Max Voting**

The max voting method is generally used for classification problems. In this technique, multiple models are used to make predictions for each data point. The predictions by each model are considered as a ‘vote’. The predictions which we get from the majority of the models are used as the final prediction.

For example, when you asked 5 of your colleagues to rate your movie (out of 5); we’ll assume three of them rated it as 4 while two of them gave it a 5. Since the majority gave a rating of 4, the final rating will be taken as 4. You can consider this as taking the mode of all the predictions.

**Averaging**

Similar to the max voting technique, multiple predictions are made for each data point in averaging. In this method, we take an average of predictions from all the models and use it to make the final prediction. Averaging can be used for making predictions in regression problems or while calculating probabilities for classification problems.

For example, in the below case, the averaging method would take the average of all the values.

i.e. (5+4+5+4+4)/5 = 4.4

**Weighted Average**

This is an extension of the averaging method. All models are assigned different weights defining the importance of each model for prediction. For instance, if two of your colleagues are critics, while others have no prior experience in this field, then the answers by these two friends are given more importance as compared to the other people.

The result is calculated as [(5\*0.23) + (4\*0.23) + (5\*0.18) + (4\*0.18) + (4\*0.18)] = 4.41.

**Libraries used in the implementation:**

NumPy: NumPy is a fundamental library for numerical computing in Python, providing support for large, multi-dimensional arrays and matrices, along with a collection of mathematical functions. It serves as a foundational tool for scientific computing tasks, enabling efficient and high-performance operations on numerical data.

Pandas: Pandas is a versatile data manipulation library in Python that offers data structures like Data Frames and Series, facilitating efficient data analysis and manipulation. It provides functionalities for cleaning, transforming, and exploring datasets, making it a go-to tool for handling structured data in various stages of the data science workflow.

Matplotlib: Matplotlib is a powerful plotting library for Python that allows the creation of diverse static, animated, and interactive visualizations. With a comprehensive set of functions, Matplotlib provides users with the flexibility to create various charts, plots, and graphs, making it an essential tool for data visualization and communication of findings.

Seaborn: Seaborn is a statistical data visualization library built on top of Matplotlib. It provides a high-level interface for creating aesthetically pleasing and informative statistical graphics. Seaborn simplifies the process of generating complex visualizations, including heatmaps, pair plots, and violin plots, while maintaining customization options for advanced users.

Metrics (Accuracy, Classification, Confusion Matrix, ROC AUC): In the context of machine learning evaluation, metrics play a crucial role. Accuracy represents the proportion of correctly classified instances, serving as a fundamental measure of model performance. Classification metrics, such as precision, recall, and F1-score, provide insights into the model's ability to correctly identify instances of a particular class. The confusion matrix presents a comprehensive summary of true positive, true negative, false positive, and false negative predictions. Lastly, the ROC AUC (Receiver Operating Characteristic - Area Under the Curve) is a performance metric for binary classification models, illustrating the trade-off between sensitivity and specificity across different thresholds, providing a holistic view of the model's discriminatory power. These metrics collectively aid in assessing and optimizing the performance of machine learning models.

**CODING**

import numpy as np

import pandas as pd

import os

for dirname, \_, filenames in os.walk('/content/drive/MyDrive/ECG'):

    for filename in filenames:

        print(os.path.join(dirname, filename))

mitbih\_test = pd.read\_csv("/content/drive/MyDrive/ECG/mitbih\_test.csv")

mitbih\_train = pd.read\_csv("/content/drive/MyDrive/ECG/mitbih\_train.csv")

ptbdb\_abnormal = pd.read\_csv("/content/drive/MyDrive/ECG/ptbdb\_abnormal.csv")

ptbdb\_normal = pd.read\_csv("/content/drive/MyDrive/ECG/ptbdb\_normal.csv")

mitbih\_test.columns = list(range(len(mitbih\_test.columns)))

mitbih\_train.columns = list(range(len(mitbih\_train.columns)))

ptbdb\_abnormal.columns = list(range(len(ptbdb\_abnormal.columns)))

ptbdb\_normal.columns = list(range(len(ptbdb\_normal.columns)))

mitbih\_test = mitbih\_test.rename({len(mitbih\_test.columns)-1: 'Label'}, axis = 1)

mitbih\_train = mitbih\_train.rename({len(mitbih\_train.columns)-1: 'Label'}, axis = 1)

ptbdb\_abnormal = ptbdb\_abnormal.rename({len(ptbdb\_abnormal.columns)-1: 'Label'}, axis = 1)

ptbdb\_normal = ptbdb\_normal.rename({len(ptbdb\_normal.columns)-1: 'Label'}, axis = 1)

(len(mitbih\_test),len(mitbih\_train),len(ptbdb\_normal),len(ptbdb\_abnormal))

data = pd.concat([mitbih\_test, mitbih\_train, ptbdb\_abnormal, ptbdb\_normal], axis = 0).sample(frac=1.0, random\_state=1).reset\_index(drop=True)

from sklearn.model\_selection import train\_test\_split

y = data['Label']

X = data.drop('Label', axis=1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, random\_state=1)

from sklearn.model\_selection import train\_test\_split

from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, classification\_report

import pandas as pd

# Initialize the Decision Tree classifier

decision\_tree\_classifier = DecisionTreeClassifier()

# Train the classifier

decision\_tree\_classifier.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = decision\_tree\_classifier.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

# Generate a classification report

clr = print(classification\_report(y\_test, y\_pred))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

                 class\_report[class\_name]['f1-score']] for class\_name in class\_names]

    # Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

                yticklabels=class\_names, cmap='GnBu')

plt.title('Classification Report Heatmap')

plt.show()

from sklearn.metrics import confusion\_matrix

cm = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm)

plt.figure(figsize=(8, 6))

sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

import pandas as pd

# Initialize the Random Forest classifier

random\_forest\_classifier = RandomForestClassifier(n\_estimators=100)  # You can adjust the number of estimators

# Train the classifier

random\_forest\_classifier.fit(X\_train, y\_train)

# Predict on the test set

y\_pred = random\_forest\_classifier.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

from sklearn.metrics import confusion\_matrix

cm1 = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm1)

plt.figure(figsize=(8, 6))

sns.heatmap(cm1, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Generate a classification report

clr1 = print(classification\_report(y\_test, y\_pred))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

                 class\_report[class\_name]['f1-score']] for class\_name in class\_names]

    # Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

                yticklabels=class\_names, cmap='GnBu')

plt.title('Classification Report Heatmap')

plt.show()

from sklearn.ensemble import VotingClassifier

ensemble\_clf = VotingClassifier(estimators=[('decision\_tree', decision\_tree\_classifier),('random\_forest', random\_forest\_classifier)], voting='hard')

# Train the ensemble classifier

ensemble\_clf.fit(X\_train, y\_train)

# Make predictions

y\_pred = ensemble\_clf.predict(X\_test)

# Calculate accuracy

accuracy = accuracy\_score(y\_test, y\_pred)

print("Ensemble Classifier Accuracy:", accuracy)

from sklearn.metrics import confusion\_matrix

cm\_ens = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm\_ens)

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_ens, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

# Generate a classification report

ens = print(classification\_report(y\_test, y\_pred))

import matplotlib.pyplot as plt

import seaborn as sns

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

                 class\_report[class\_name]['f1-score']] for class\_name in class\_names]

    # Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

                yticklabels=class\_names, cmap='GnBu')

plt.title('Classification Report Heatmap')

**FRAMEWORK CODING:**

import tkinter as tk

import tkinter as tk

from tkinter import ttk

from sklearn.ensemble import VotingClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

from sklearn.metrics import roc\_auc\_score, roc\_curve, auc, precision\_recall\_fscore\_support

import seaborn as sns

import matplotlib.pyplot as plt

from matplotlib.backends.backend\_tkagg import FigureCanvasTkAgg

from PIL import Image, ImageTk

from sklearn.model\_selection import train\_test\_split

import numpy as np

import pandas as pd

import os

for dirname, \_, filenames in os.walk('D:\varsha\ECG'):

for filename in filenames:

print(os.path.join(dirname, filename))

mitbih\_test = pd.read\_csv("mitbih\_test.csv")

mitbih\_train = pd.read\_csv("mitbih\_train.csv")

ptbdb\_abnormal = pd.read\_csv("ptbdb\_abnormal.csv")

ptbdb\_normal = pd.read\_csv("ptbdb\_normal.csv")

mitbih\_test.columns = list(range(len(mitbih\_test.columns)))

mitbih\_train.columns = list(range(len(mitbih\_train.columns)))

ptbdb\_abnormal.columns = list(range(len(ptbdb\_abnormal.columns)))

ptbdb\_normal.columns = list(range(len(ptbdb\_normal.columns)))

mitbih\_test = mitbih\_test.rename({len(mitbih\_test.columns)-1: 'Label'}, axis = 1)

mitbih\_train = mitbih\_train.rename({len(mitbih\_train.columns)-1: 'Label'}, axis = 1)

ptbdb\_abnormal = ptbdb\_abnormal.rename({len(ptbdb\_abnormal.columns)-1: 'Label'}, axis = 1)

ptbdb\_normal = ptbdb\_normal.rename({len(ptbdb\_normal.columns)-1: 'Label'}, axis = 1)

(len(mitbih\_test),len(mitbih\_train),len(ptbdb\_normal),len(ptbdb\_abnormal))

data = pd.concat([mitbih\_test, mitbih\_train, ptbdb\_abnormal, ptbdb\_normal], axis = 0).sample(frac=1.0, random\_state=1).reset\_index(drop=True)

y = data['Label']

X = data.drop('Label', axis=1)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, train\_size=0.7, random\_state=1)

# Initialize classifiers

decision\_tree\_classifier = DecisionTreeClassifier()

random\_forest\_classifier = RandomForestClassifier(n\_estimators=100)

ensemble\_clf = VotingClassifier(estimators=[('decision\_tree', decision\_tree\_classifier),('random\_forest', random\_forest\_classifier)], voting='hard')

# Tkinter GUI

root = tk.Tk()

root.title("Classifier Metrics")

root.geometry("400x400")

# Load background image

background\_image = Image.open("b1.jpg") # Replace with your image file

background\_photo = ImageTk.PhotoImage(background\_image)

background\_label = tk.Label(root, image=background\_photo)

background\_label.place(relwidth=1, relheight=1)

# Project label

project\_label = tk.Label(root, text="Heart Disease Prediction using hybrid learning models for multimodal ECG Dataset", font=("Helvetica", 12), bg="white")

project\_label.pack(pady=10)

# Labels for dataset information

r\_dataset\_label = tk.Label(root, text="Dataset: ECG", font=("Helvetica", 11),foreground="blue",width=20)

r\_dataset\_label.pack(pady=10, padx=10)

# Training Data Label

r\_train\_data\_label = tk.Label(root, text="Training Data: 70%", font=("Helvetica", 11),foreground="blue",width=20)

r\_train\_data\_label.pack(pady=10, padx=10)

# Testing Data Label

r\_test\_data\_label = tk.Label(root, text="Testing Data: 30%", font=("Helvetica", 11), foreground="blue",width=20)

r\_test\_data\_label.pack(pady=10, padx=10)

# Function to train classifiers

def train\_dtc\_classifier():

global decision\_tree\_classifier, X\_train, y\_train

decision\_tree\_classifier.fit(X\_train, y\_train)

print("DTC Classifier trained successfully.")

def train\_rfc\_classifier():

global random\_forest\_classifier, X\_train, y\_train

random\_forest\_classifier.fit(X\_train, y\_train)

print("RFC Classifier trained successfully.")

def train\_ensemble\_classifier():

global ensemble\_clf, X\_train, y\_train

ensemble\_clf.fit(X\_train, y\_train)

print("Ensemble Classifier trained successfully.")

# Function to calculate metrics and show charts for DTC

def show\_dtc\_metrics():

global decision\_tree\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = decision\_tree\_classifier.predict(X\_test)

# Confusion Matrix

cm\_dtc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm\_dtc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_dtc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of dtc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_dtc():

# Predict the Test set results

y\_pred = decision\_tree\_classifier.predict(X\_test)

# Classification Report

clr\_dtc = print(classification\_report(y\_test, y\_pred))

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of dtc')

plt.show()

def calculate\_accuracy\_dtc():

global decision\_tree\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = decision\_tree\_classifier.predict(X\_test)

# Accuracy

accuracy\_dtc = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of dtc:', accuracy\_dtc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_dtc], color='blue')

plt.title('Model Accuracy of dtc')

plt.ylabel('Accuracy')

plt.show()

# Function to calculate metrics and show charts for RFC

def show\_rfc\_metrics():

global random\_forest\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = random\_forest\_classifier.predict(X\_test)

# Confusion Matrix

cm\_rfc = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix\n\n', cm\_rfc)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_rfc, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of rfc')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_rfc():

# Predict the Test set results

y\_pred = random\_forest\_classifier.predict(X\_test)

# Classification Report

clr\_rfc = print(classification\_report(y\_test, y\_pred))

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of rfc')

plt.show()

def calculate\_accuracy\_rfc():

global random\_forest\_classifier, X\_test, y\_test

# Predict the Test set results

y\_pred = random\_forest\_classifier.predict(X\_test)

# Accuracy

accuracy\_rfc = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of rfc:', accuracy\_rfc)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_rfc], color='blue')

plt.title('Model Accuracy of rfc')

plt.ylabel('Accuracy')

plt.show()

# Function to calculate metrics and show charts for SVM

def show\_ensemble\_metrics():

global ensemble\_clf, X\_test, y\_test

# Predict the Test set results

y\_pred = ensemble\_clf.predict(X\_test)

# Confusion Matrix

cm\_ens = confusion\_matrix(y\_test, y\_pred)

print('Confusion matrix of ensemble\n\n', cm\_ens)

# Plot Confusion Matrix

plt.figure(figsize=(8, 6))

sns.heatmap(cm\_ens, annot=True, fmt='d', cmap='Blues', cbar=False)

plt.title('Confusion Matrix of Ensemble')

plt.xlabel('Predicted')

plt.ylabel('True')

plt.show()

def show\_report\_ens():

# Predict the Test set results

y\_pred = ensemble\_clf.predict(X\_test)

# Classification Report

clr\_ens = print(classification\_report(y\_test, y\_pred))

# Plot Classification Report

class\_report = classification\_report(y\_test, y\_pred, output\_dict=True)

class\_names = [str(label) for label in class\_report.keys() if label not in ['accuracy', 'macro avg', 'weighted avg']]

heatmap\_data = [[class\_report[class\_name]['precision'], class\_report[class\_name]['recall'],

class\_report[class\_name]['f1-score']] for class\_name in class\_names]

# Create a heatmap

fig, ax = plt.subplots(figsize=(10, 6))

sns.heatmap(heatmap\_data, annot=True, fmt=".2f", xticklabels=['Precision', 'Recall', 'F1-Score'],

yticklabels=class\_names, cmap='Blues')

plt.title('Classification Report Heatmap of ensemble')

plt.show()

def calculate\_accuracy\_ens():

global ensemble\_clf, X\_test, y\_test

# Predict the Test set results

y\_pred = ensemble\_clf.predict(X\_test)

# Accuracy

accuracy\_ens = accuracy\_score(y\_test, y\_pred)

print('Model accuracy score of ensemble:', accuracy\_ens)

# Plot Accuracy

plt.figure(figsize=(6, 4))

plt.bar(["Accuracy"], [accuracy\_ens], color='blue')

plt.title('Model Accuracy of Ensemble')

plt.ylabel('Accuracy')

plt.show()

# DTC Frame

dtc\_frame = tk.Frame(root)

dtc\_frame.pack(side=tk.TOP, pady=10)

# DTC Train Button

dtc\_train\_button = tk.Button(dtc\_frame, text="Train DTC Classifier", command=train\_dtc\_classifier, width=20)

dtc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Metrics Button

dtc\_metrics\_button = tk.Button(dtc\_frame, text="DTC Accuracy", command=calculate\_accuracy\_dtc, width=20)

dtc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_matrix\_button = tk.Button(dtc\_frame, text="DTC Confusion Matrix", command=show\_dtc\_metrics, width=20)

dtc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# DTC Matrix Button

dtc\_report\_button = tk.Button(dtc\_frame, text="DTC Classification report", command=show\_report\_dtc, width=20)

dtc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Frame

rfc\_frame = tk.Frame(root)

rfc\_frame.pack(side=tk.TOP, pady=10)

# RFC Train Button

rfc\_train\_button = tk.Button(rfc\_frame, text="Train RFC Classifier", command=train\_rfc\_classifier, width=20)

rfc\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Metrics Button

rfc\_metrics\_button = tk.Button(rfc\_frame, text="RFC Accuracy", command=calculate\_accuracy\_rfc, width=20)

rfc\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC Matrix Button

rfc\_matrix\_button = tk.Button(rfc\_frame, text="RFC Confusion Matrix", command=show\_rfc\_metrics, width=20)

rfc\_matrix\_button.pack(side=tk.LEFT, padx=5, pady=5)

# RFC report Button

rfc\_report\_button = tk.Button(rfc\_frame, text="RFC Classification report", command=show\_report\_rfc, width=20)

rfc\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Ensemble Frame

ens\_frame = tk.Frame(root)

ens\_frame.pack(side=tk.TOP, pady=10)

# Ensemble Train Button

ens\_train\_button = tk.Button(ens\_frame, text="Train Ensemble Classifier", command=train\_ensemble\_classifier, width=20)

ens\_train\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Ensemble Metrics Button

ens\_metrics\_button = tk.Button(ens\_frame, text="Ensemble Accuracy", command=calculate\_accuracy\_ens, width=20)

ens\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Ensemble matrix Button

ens\_metrics\_button = tk.Button(ens\_frame, text="Ensemble Confusion Matrix", command=show\_ensemble\_metrics, width=20)

ens\_metrics\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Ensemble report Button

ens\_report\_button = tk.Button(ens\_frame, text="Ensemble Classification report", command=show\_report\_ens, width=20)

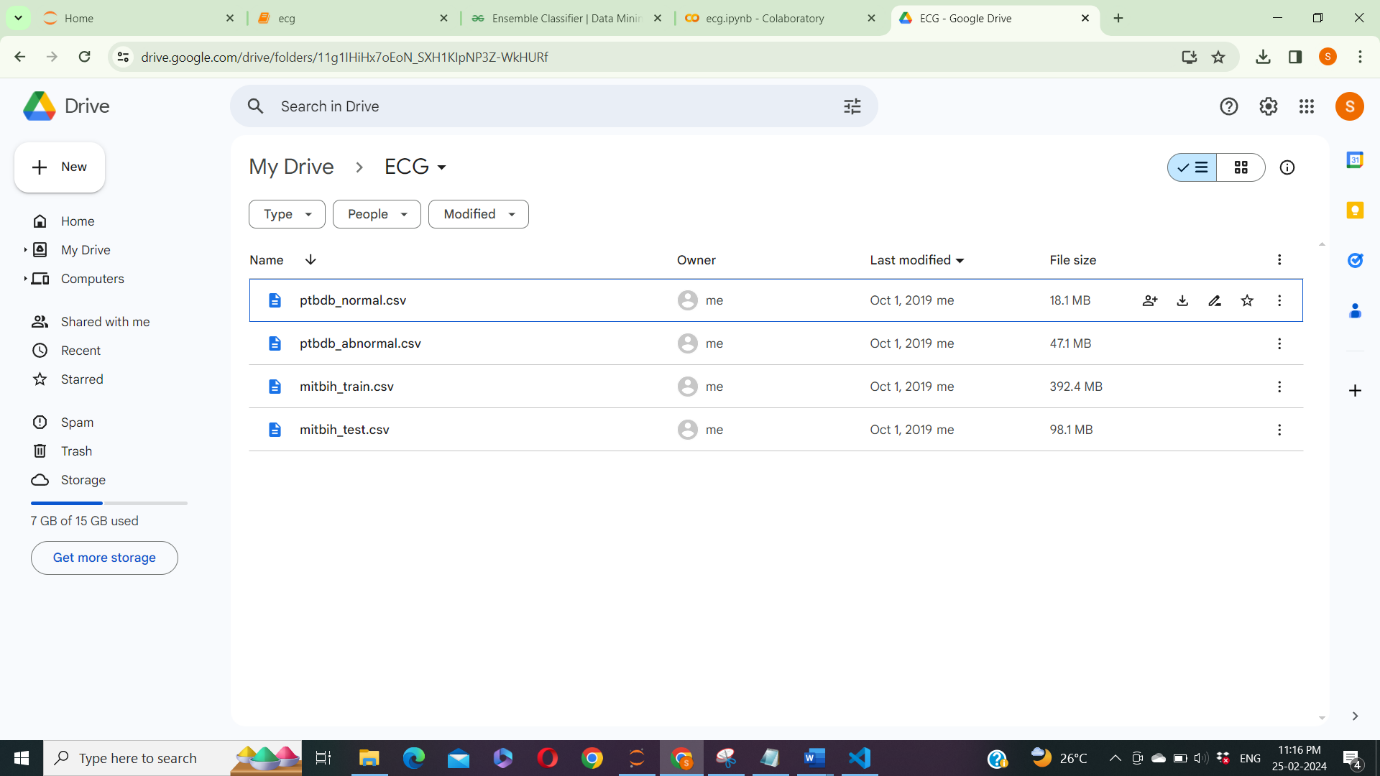
ens\_report\_button.pack(side=tk.LEFT, padx=5, pady=5)

# Run the Tkinter event loop

root.mainloop()

**RESULTS AND DISCUSSION:**

**Dataset:**

Figure 1: CSV dataset

**Results:**

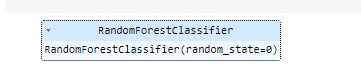


Figure 2: Random forest classifier algorithm



Figure 3: Accuracy calculation of Random Forest classifier algorithm

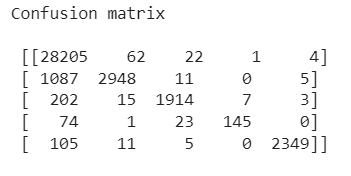


Figure 4: Confusion matrix calculation of Random Forest classifier algorithm

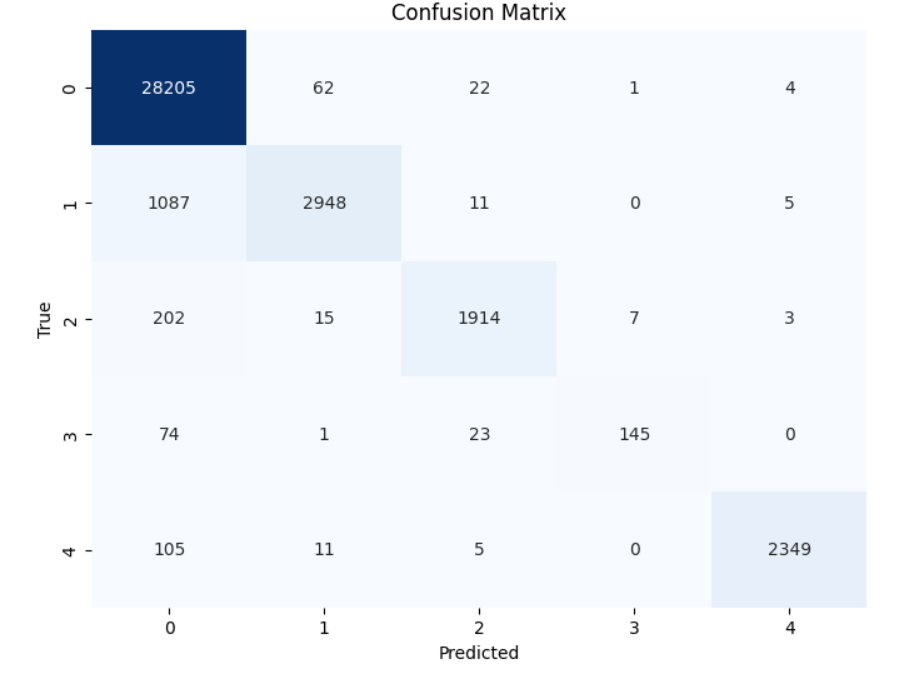


Figure 5: Confusion matrix graph of Random Forest classifier algorithm

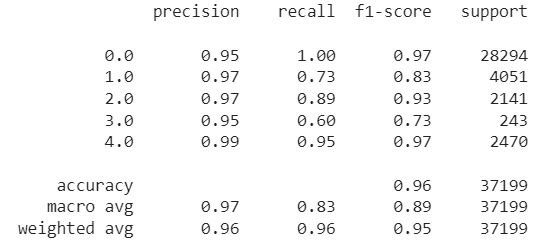


Figure 6: Classification report calculation of Random Forest classifier algorithm

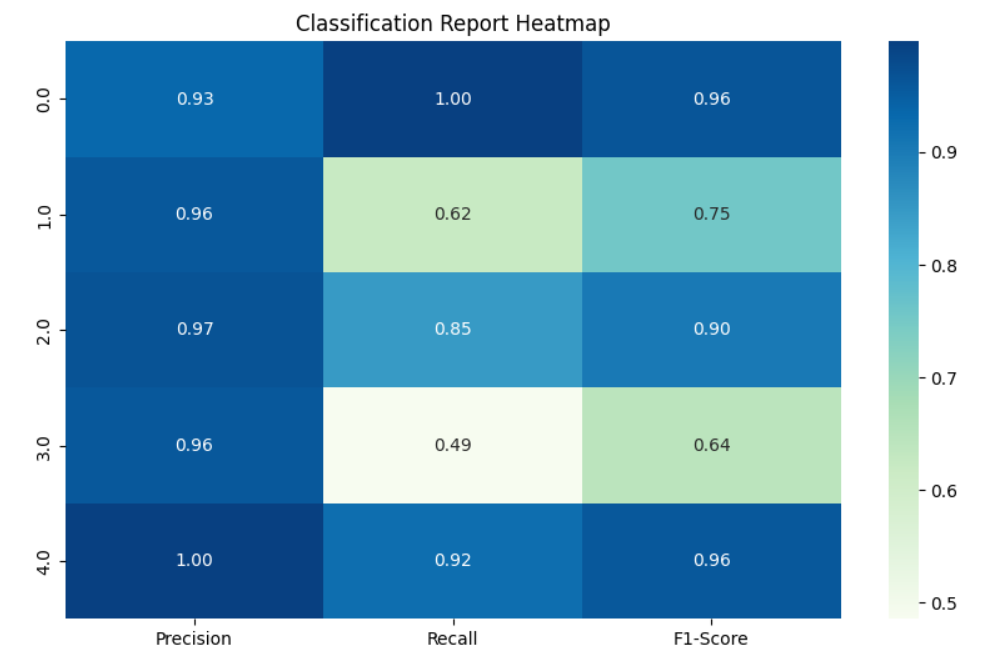


Figure 7: Classification report graph of Random Forest classifier algorithm

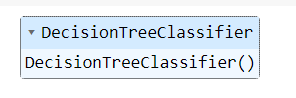


Figure 8: Decision Tree classifier algorithm



Figure 9: Accuracy calculation of Decision Tree classifier algorithm

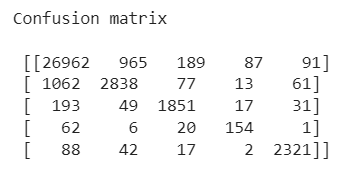


Figure 10: Confusion matrix of Decision Tree classifier algorithm

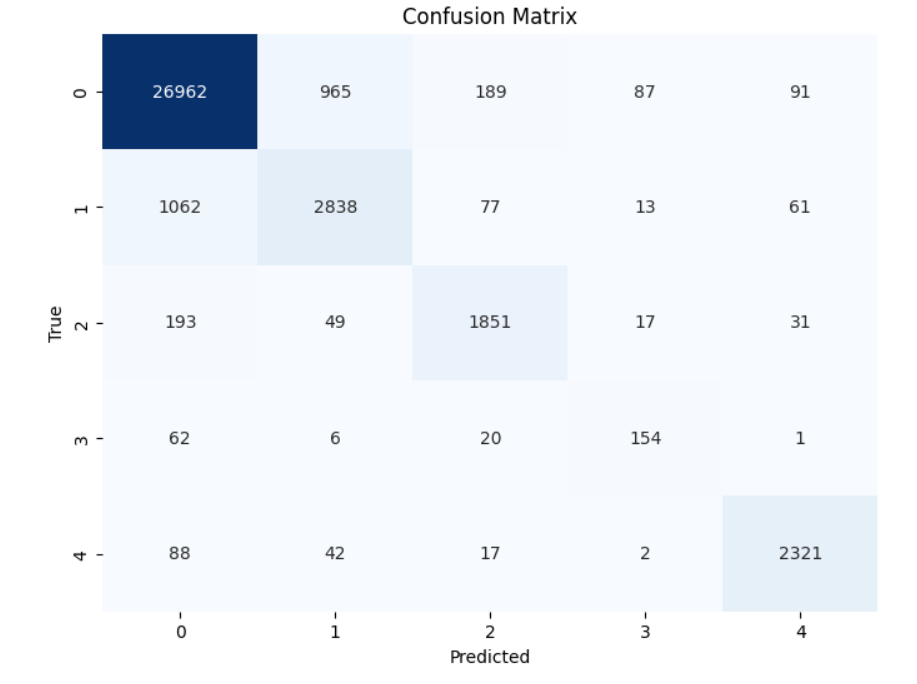


Figure 11: Confusion matrix graph of Decision Tree classifier algorithm

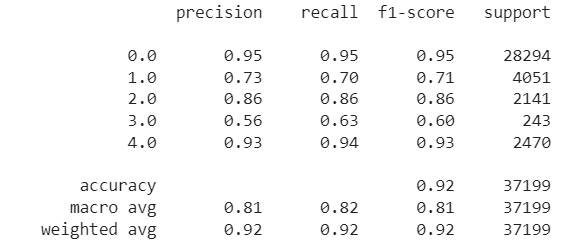


Figure 12: Classification report of Decision Tree classifier algorithm

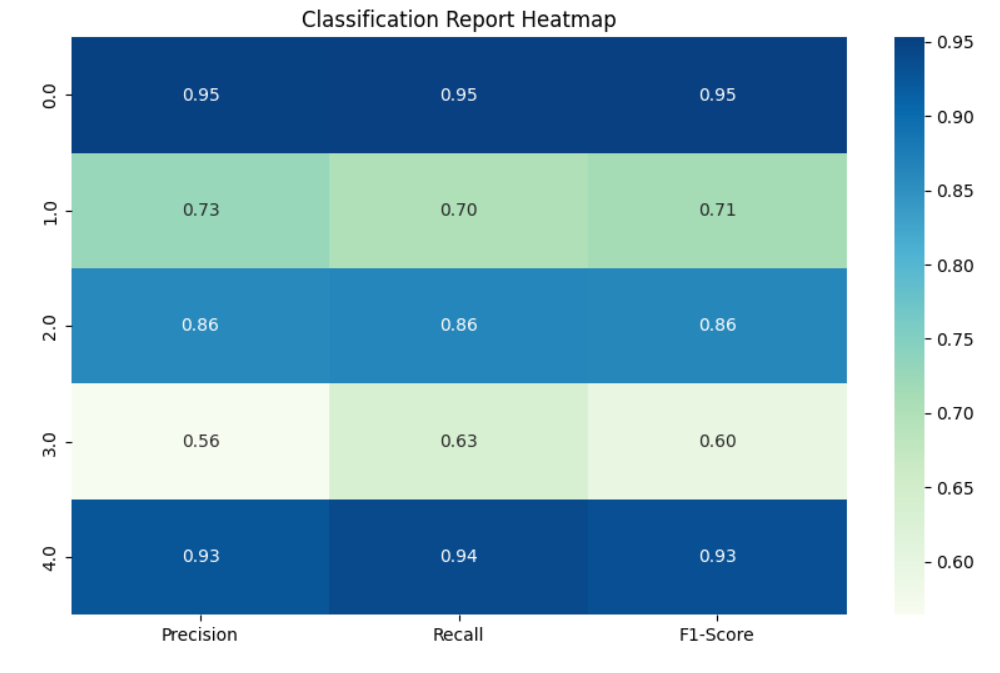


Figure 13: Classification report graph of Decision Tree classifier algorithm

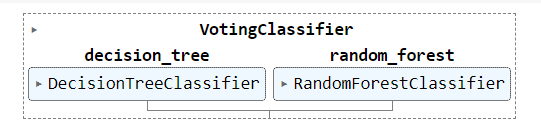


Figure 14: Ensemble Classifier Algorithm



Figure 15: Accuracy Calculation of Ensemble Classifier Algorithm

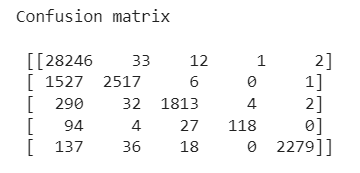


Figure 16: Confusion Matrix of Ensemble Classifier Algorithm

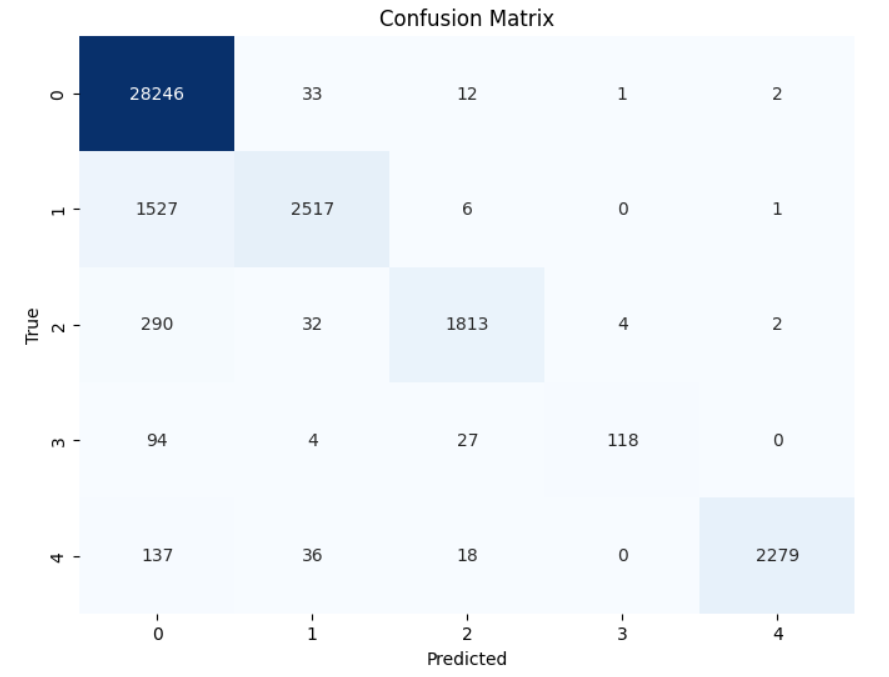


Figure 17: Confusion Matrix graph of Ensemble Classifier Algorithm

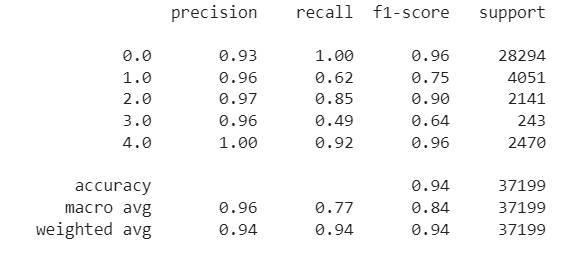


Figure 18: Classification report of Ensemble Classifier Algorithm

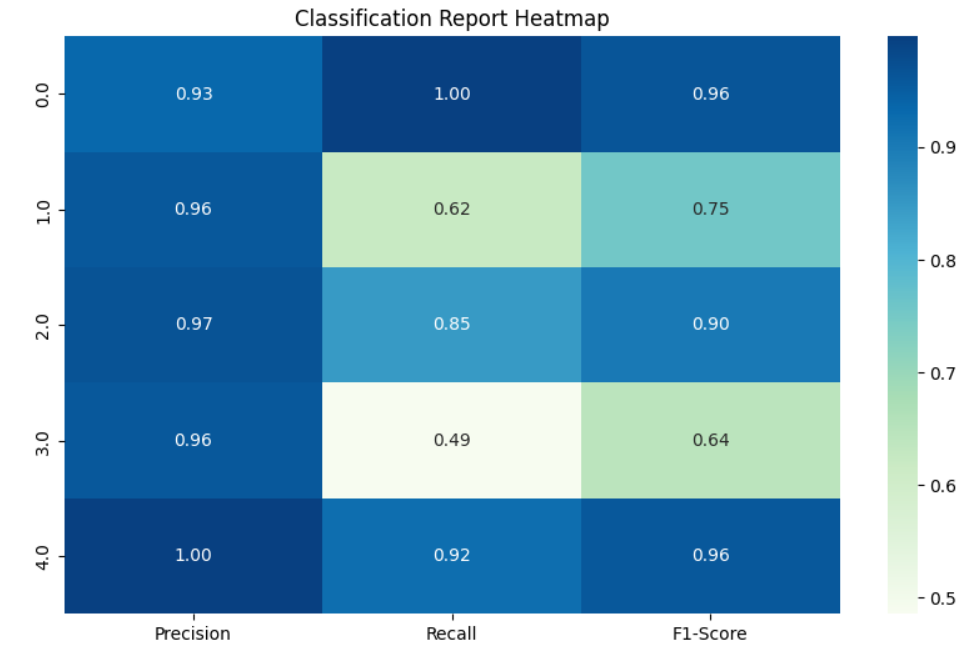


Figure 19: Classification report graph of Ensemble Classifier Algorithm

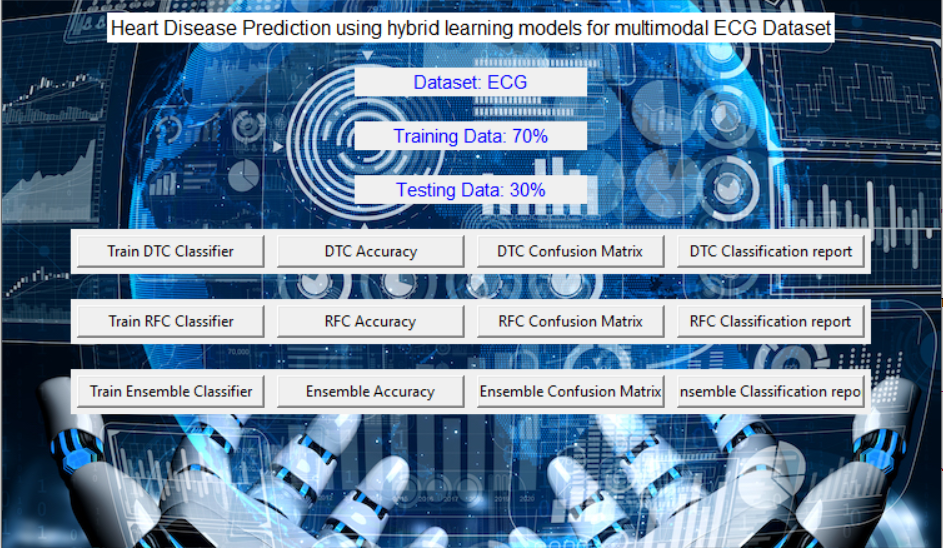


Figure 20: Frame work design







Figure 21: Classifier training

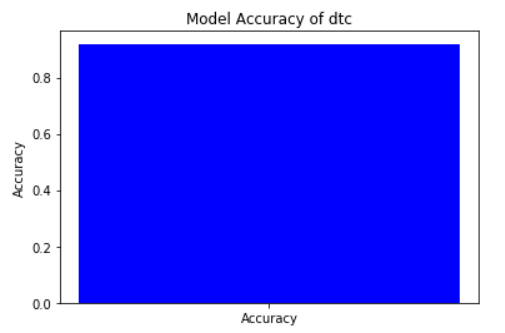


Figure 22: Accuracy graph of Decision Tree classifier algorithm

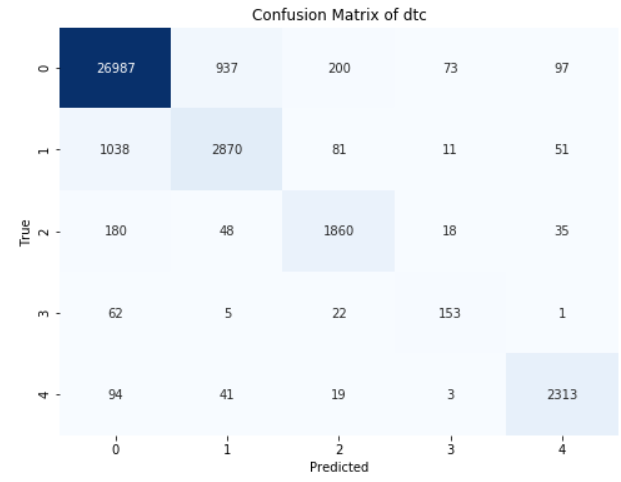


Figure 23: Confusion matrix graph of Decision Tree classifier algorithm

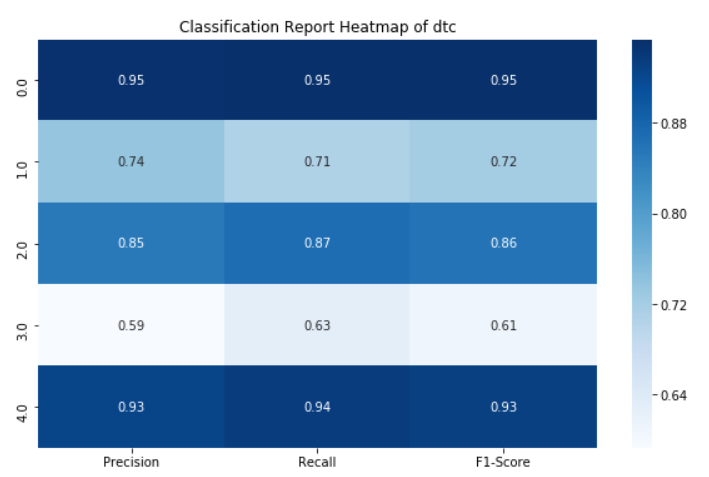


Figure 24: Classification report graph of Decision Tree classifier algorithm

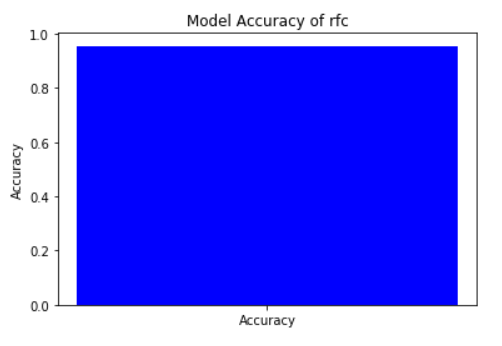


Figure 25: Accuracy graph of Random Forest classifier algorithm

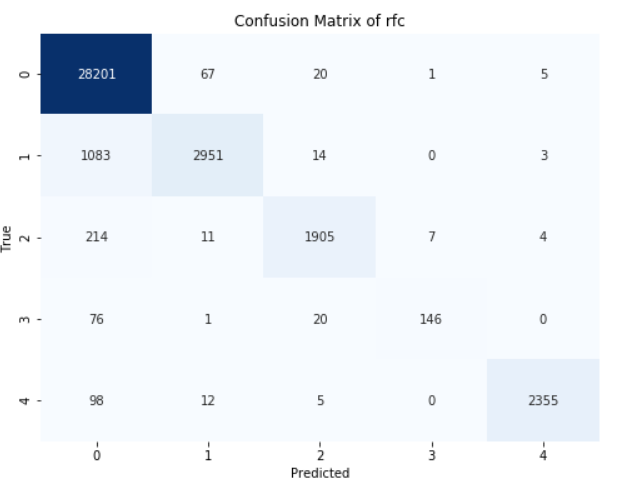


Figure 26: Confusion matrix graph of Random Forest classifier algorithm

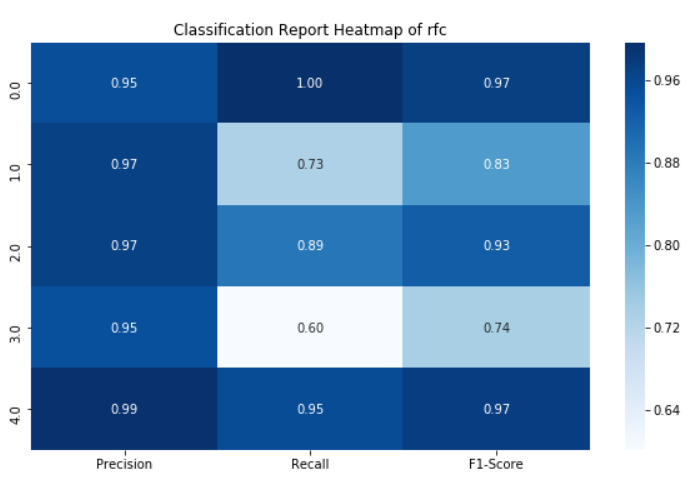


Figure 27: Classification report graph of Random Forest classifier algorithm

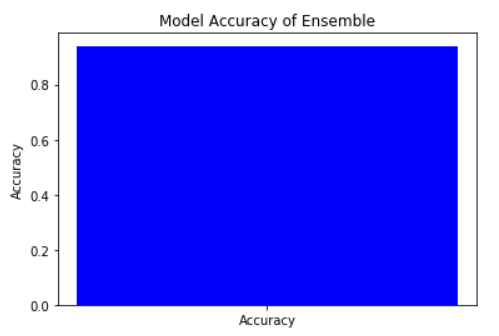


Figure 28: Accuracy graph of Ensemble classifier algorithm

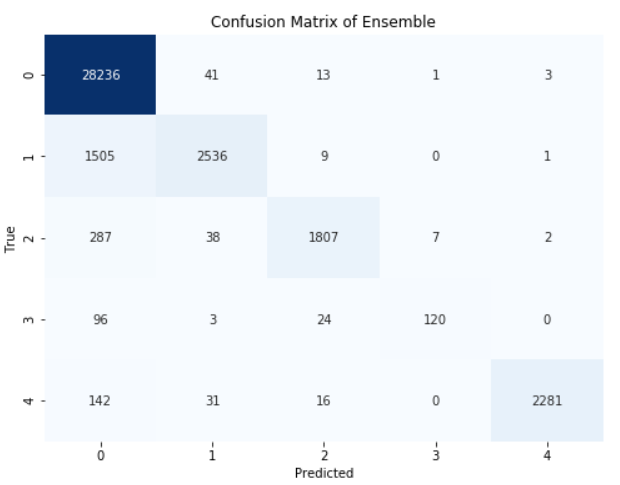


Figure 29: Confusion matrix graph of Ensemble classifier algorithm

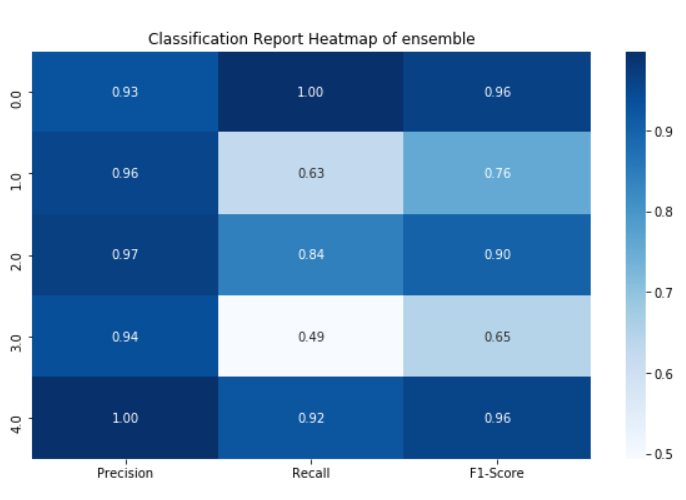


Figure 30: Classification report graph of Ensemble classifier algorithm

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 – score | Support |
| 0.0 | 0.95 | 0.95 | 0.95 | 28294 |
| 1.0 | 0.73 | 0.70 | 0.71 | 4051 |
| 2.0 | 0.86 | 0.86 | 0.86 | 2141 |
| 3.0 | 0.56 | 0.63 | 0.60 | 243 |
| 4.0 | 0.93 | 0.94 | 0.93 | 2470 |
| accuracy |  |  | 0.92 | 37199 |
| Macro avg | 0.81 | 0.82 | 0.81 | 37199 |
| Weighted avg | 0.92 | 0.92 | 0.92 | 37199 |

Table 1: classification report of DTC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 – score | Support |
| 0.0 | 0.95 | 1.00 | 0.97 | 28294 |
| 1.0 | 0.97 | 0.73 | 0.83 | 4051 |
| 2.0 | 0.97 | 0.89 | 0.93 | 2141 |
| 3.0 | 0.95 | 0.60 | 0.73 | 243 |
| 4.0 | 0.99 | 0.95 | 0.97 | 2470 |
| accuracy |  |  | 0.96 | 37199 |
| Macro avg | 0.97 | 0.83 | 0.89 | 37199 |
| Weighted avg | 0.96 | 0.96 | 0.95 | 37199 |

Table 2: classification report of RFC

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Precision | Recall | F1 – score | Support |
| 0.0 | 0.93 | 1.00 | 0.96 | 28294 |
| 1.0 | 0.96 | 0.62 | 0.75 | 4051 |
| 2.0 | 0.97 | 0.85 | 0.90 | 2141 |
| 3.0 | 0.96 | 0.49 | 0.64 | 243 |
| 4.0 | 1.00 | 0.92 | 0.96 | 2470 |
| accuracy |  |  | 0.94 | 37199 |
| Macro avg | 0.96 | 0.77 | 0.84 | 37199 |
| Weighted avg | 0.94 | 0.94 | 0.94 | 37199 |

Table 3: Classification Report of Ensemble classifier

The classification report is a performance evaluation tool that shows the precision, recall, f1-score, for each class in a classification problem. In training images using the deep learning model, the classification report would provide information about how well the model performed in classifying images into different categories. The precision represents the percentage of correctly classified images among all the images classified as belonging to a specific class. The recall represents the percentage of correctly classified images among all the images that actually belong to a specific class. The f1-score is a harmonic mean of precision and recall, and support represents the number of images in each class.

The accuracy has been calculated for the model that has been implemented, and the result for the model is compared in Table

|  |  |
| --- | --- |
| Algorithms | Accuracy |
| DTC | 91 |
| RFC | 95 |
| Ensemble Classifier | 94 |

Table 4: Accuracy comparison of algorithm.

|  |  |  |
| --- | --- | --- |
| Dataset Count | Training Value | Testing Value |
| 123998 | 70 | 30 |

Table 4: Consist of dataset count, Training and Testing percentage.

**CONCLUSION**

In conclusion, the application of hybrid learning techniques to multimodal ECG datasets for heart disease prediction presents a promising avenue for advancing clinical decision-making and patient care. By leveraging the diverse information captured by different modalities within ECG recordings, such as waveform morphology, heart rate variability, and other relevant parameters, hybrid learning models can effectively integrate complementary insights to improve prediction accuracy. Through the combination of decision tree, random forest, and ensemble classifier algorithms, the proposed approach harnesses the strengths of each method while mitigating their individual limitations, resulting in robust and reliable predictions. The comprehensive evaluation of model performance using metrics such as accuracy, precision, recall, and confusion matrices provides valuable insights into the effectiveness of the hybrid learning approach. Ultimately, the adoption of such methodologies holds significant potential for enhancing the early detection, diagnosis, and management of heart disease, thereby contributing to improved patient outcomes and healthcare delivery. Continued research and refinement of hybrid learning techniques in conjunction with multimodal ECG datasets are essential for realizing the full potential of this approach in clinical practice.

**REFERNCES**

Attia, Ziad I., et al. "Multimodal deep learning for cardiovascular risk prediction." Journal of the American Heart Association 8.7 (2019): e011638.

Ayres, Gustavo, et al. "Cardiovascular disease prediction using ensemble learning on ECG signals." 2016 38th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). IEEE, 2016.

Gomes, Paulo, et al. "Combining CNN and RNN for heart disease prediction from ECG signals." 2018 IEEE EMBS International Conference on Biomedical & Health Informatics (BHI). IEEE, 2018.

Kachuee, Mohammad, Shayan Fazeli, and Majid Sarrafzadeh. "ECG heartbeat classification: A deep transferable representation." 2018 40th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC). IEEE, 2018.

Maji, Suman Kumar, et al. "Heart disease prediction from ECG signals using deep learning and machine learning algorithms." 2019 IEEE Calcutta Conference (CALCON). IEEE, 2019.

Ramachandran, Nirmal, and Sujatha Krishnamoorthy. "Hybrid machine learning model for prediction of heart disease using ECG signals." 2019 IEEE International Conference on Intelligent Techniques in Control, Optimization and Signal Processing (INCOS). IEEE, 2019.

Rashid, Mohammad Mamunur, et al. "Deep learning for heart disease diagnosis: A review." Journal of Ambient Intelligence and Humanized Computing 11.1 (2020): 349-361.