SUPERVISED MACHINE LEARNING FOR
HEART DISEASE PREDICTION

Report submitted to the SASTRA Deemed to be University
as the requirement for the course

INT300: MINI PROJECT

Submitted by

RATHISH PANDIAN
(Reg. No.: 124015077, B.Tech. Information Technology)
SHYAM SUNDAR G S
(Reg. No.: 124015145, B.Tech. Information Technology)
PRANESH VJ
(Reg. No.: 124015155, B.Tech. Information Technology)

May 2023

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

Bonafide Certificate

This is to certify that the report titled "**Supervised Machine Learning For Heart Disease Prediction**" submitted as a requirement for the course, **INT300: MINI PROJECT** for B.Tech. is a bonafide record of the work done by **Mr. Rathish Pandian (Reg. No. 124015077, B.Tech. Information Technology**), **Mr. Shyam Sundar G S (Reg. No. 124015145, B.Tech. Information Technology**) and **Mr. Pranesh VJ (Reg. No. 124015155, B.Tech. Information Technology**) during the academic year 2022-23, in the School of Computing, under my supervision.

Signature of Project Supervisor: Ramakrishnan

Name with Affiliation: Dr. Ramakrishnan S, Assistant Professor, SOC

Date: 15/05/2023

Mini Project *Viva voce* held on ___________________________

Examiner 1 **Examiner 2**

Acknowledgements

We would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as part of our curriculum.

We would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam and Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

We extend our sincere thanks **to Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

We extend our heartfelt thanks to **Dr. A. Umamakeswari**, Dean, School of Computing**, Dr. S. Gopalakrishnan**, Associate Dean, Department of Computer Application, **Dr. B.Santhi**, Associate Dean, Research, **Dr. V. S. Shankar Sriram**, Associate Dean, Department of Computer Science and Engineering, **Dr. R. Muthaiah**, Associate Dean, Department of Information Technology, and Information & Communication Technology

Our guide **Dr. Ramakrishnan S**, Assistant Professor, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped me/us in making progress throughout our project work. We also thank the project review panel members for their valuable comments and insights which made this project better.

We would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

We gratefully acknowledge all the contributions and encouragement from our family and friends resulting in the successful completion of this project. We thank you all for providing mean opportunity to showcase our skills through project.

List of Figures

List of Tables

ABBREVIATIONS

ML – Machine Learning

CVD – Cardio Vascular Diseases

UCI – University of California, Irvine

EDA – Exploratory Data Analysis

ABM1 – AdaBoostM1

KNN – K-Nearest Neighbors

NB – Naïve Bayes

DT – Decision Tree

RF – Random Forest

HRFLM – Hybrid Random Forest with a Linear Model

LR – Logistic Regression

SVM – Support Vector Machine

MLP – Multi Layer Perceptron

CNN – Convolutional Neural Network

SMOTE – Synthetic Minority Oversampling Technique

KDE – Kernel Density Estimation

IQR – Inter Quartile Range

TP - True Positive

TN - True Negative

FP - False Positive

FN – False Negative

TPR - True Positive Rate

FPR - False Positive Rate

MCC – Matthew’s Correlation Co-efficient

AUROC – Area Under Receiver Operating Characteristic Curve

AUPRC – Area Under Precision Recall Curve

ABSTRACT

There is a proliferation of deaths due to cardiovascular diseases in the modern world . Many nations lack enough cardiovascular expertise, which has resulted in a considerable proportion of instances receiving wrong diagnoses. This project aims at implementing machine learning classifiers for these diagnostic uses. We performed this project on a heart disease dataset collected from Kaggle. We implemented six machine learning algorithms namely Logistic Regression, AdaBoostM1 (ABM1), Random Forest (RF), K-Nearest Neighbor (KNN), Decision Tree (DT), Multilayer Perceptron (MLP). KNN performed extremely well giving us 100% accuracy. ABM1, DT and RF also produced good results with accuracies in the range 99-100%. While MLP gave an accuracy of 95.175%, LR gave the least accuracy with a value of 86.271%.

KEY WORDS: Machine Learning, Cardiovascular Disease Prediction, Supervised Machine Learning.

Table of Contents

CHAPTER-1

SUMMARY OF BASE PAPER

Title: Heart disease prediction using supervised machine learning algorithms Performance analysis and comparison. [2]

Year of Publication: 2022

Journal Name: Computers in Biology and Medicine

Indexed in: Science Citation Index Expanded (SCIE)

URL: : https://www.sciencedirect.com/science/article/pii/S0010482521004662

1.1 INTRODUCTION:

Over the past few years, there has been a proliferation of mortalities due to cardiovascular diseases. Most of these mortalities happen because of the failure to predict the CVD at an initial stage. In fact, many countries find it difficult to afford the modern equipment required for the discovery of these diseases. Machine Learning methods can be used as a cheaper alternative as they have been very effective in making decisions and predictions in the healthcare industry over the years.

1.2 METHODOLOGY:

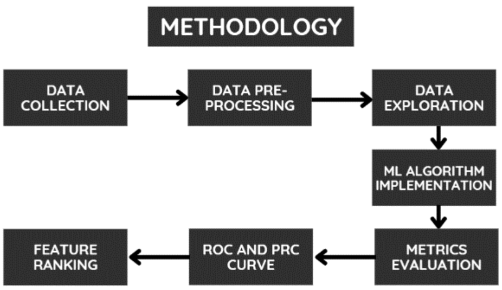


Figure 1.1 – Methodology

1.2.1 Data Collection:

We performed this project on a heart disease dataset collected from Kaggle [1]. The dataset contains 1025 samples and 14 attributes. Table 1.1 depicts the features of each attribute mentioned in the dataset.

1.2.2 Data Pre-processing:

It is important to remove outliers from the dataset as they might affect the outcome of our machine learning model. The Inter Quartile Range (IQR) method was used to remove the outliers from the given dataset. Following the removal of the outliers, the imbalanced dataset was balanced using Synthetic Minority Oversampling Technique (SMOTE).

Attribute	Description
age	Age of the patients in Years
sex	Male=1;Female=0;
cp	Type of Chest Pain (4 types)
trestbps	Resting bp (blood pressure) in mmHg
chol	Serum cholesterol measured in mg/dl
fbs	Fasting blood sugar > 120 mg/dl (true =1; false =0)
restecg	Resting electrocardiograph results
thalach	Maximum heart rate discovered
exang	Exercise induced angina (yes=1; no=0)
oldpeak	ST depression induced by exercise relative to rest
slope	Slope of peak exercise ST segment
ca	No. of major vessels (0-3)
thal	Normal=1; Fixed defect=2; Reversible Defect=3;
target	No disease=0; Disease=1;

Table 1.1 – Attribute Description

1.2.3 Data Exploration:

A Heatmap showing correlation among the attributes was plotted to understand the correlation between various attributes. A Kernel Density Estimation (KDE) plot was also plotted to understand the distribution of diseased and non-diseased samples based on the age.

1.2.4 Machine Learning Algorithms Implementation

Six supervised Machine Learning algorithms were implemented in the project. 10-fold cross-validation was implemented to train and test the models. In 10-fold cross validation, the data is partitioned into 10 folds where 9 folds are used for training the model and the remaining fold is used for testing purpose. The test and train splits are changed for each validation. GridSearchCV was used for hyperparameter tuning to find out the best parameters for the models.

K- Nearest Neighbors (KNN):

KNN is not a parametric classification algorithm, rather it is a type of non-parametric classification algorithm which means the algorithm does not make any assumptions about the distribution of the data. In KNN, the test data is classified based on the nearest k neighbors in the training data. When we want to identify the class of the test sample, the distance between its k-neighbors is calculated to classify the sample. Here, K refers to the number of neighbors that must be considered for predicting the class of the test sample.

Logistic Regression (LR):

Unlike the name suggests, Logistic Regression is used for classification rather than regression. LR finds the relationship between the target attribute and the other attributes based on the probability value calculated using a logistic function whose value always lies between 0 and 1. We can also set a threshold value to classify the samples where values upwards of the threshold are changed as 1 while values downward of the threshold are changed to 0.

Multi-Layer Perceptron (MLP):

Multilayer Perceptron is a neural network-based algorithm used for classification. The basic building block of MLP is neuron. Each layer has multiple neurons which inter-connects multiple layers. It usually has three or more layers. The Input layer accepts the input and forwards it to hidden layer. The hidden layer transforms the input received to the output layer by performing some computation using non-linear activation functions. There can be multiple hidden layers but only one input and output layer. In case of error in the output layer, the error is back propagated till there is no significant error in the output.

Decision Tree (DT):

Like KNN, Decision Tree is also a non-parametric algorithm. As the name suggests it has a tree like structure with root node and leaf node. The root node is used to make the decision and can be branched further while leaf nodes cannot be branched further. When we want to classify a test sample, we start right from the topmost node which is the root node and go till leaf node to predict the class. We compare the attributes of the test data with the root attribute and follow the branch correspondingly to classify the sample.

Random Forest(RF):

Like ABM1, RF is also an ensemble approach. As the name suggests, Random Forest contains a forest of trees i.e., multiple number of decision trees are created resulting in a forest. Every Decision tree will predict the class label during the training phase. When we want to classify a test sample, the class that obtains the most votes among the decision tree are assigned as the class label for that sample.

AdaBoostM1 (ABM1):

ABM1, an extension of the AdaBoost algorithm, is an ensemble learning technique where multiple weak classifiers are integrated into a strong classifier to obtain better results based on adaptive enhancement approach. Decision Tree was used as the base classifier. A base classifier is trained on the input data, and the misclassified instances are assigned more weight. The process is iterated for a fixed number where base classifier changes for every instance based on the prior misclassified instances. We calculate the weighted sum to make the final prediction.

1.2.5 Metrics Evaluation:

After implementing our model for classification, we calculate various metrics to find out the performance of our model. The metrics we calculate include:

Accuracy: Percentage of correctly identified predictions with respect to test data.

Precision: Out of all the predicted positives, how much true positives are predicted.

Recall: Out of all positives, how many true positives have been predicted.

F1-Score: Used to find the trade-off/balance between precision and recall.

Sensitivity: Indicates how many true positives are identified out of all positives.

Specificity: Indicates how many true negatives are identified out of all negatives.

Kappa score: Indicates the agreement between predicted and test labels of our model considering the agreement that can occur by chance.

MCC score: Indicates the quality of binary classifications considering the predicted and test labels.

AUROC: Gives area under ROC curve.

AUPRC: Gives area under the PRC curve.

1.2.6 ROC AND PRC Curve Analysis:

Based on the AUROC and AUPRC values obtained, ROC and PRC curves were plotted for different models. Receiver Operating Characteristic (ROC) curve shows the trade-off between specificity and sensitivity measuring the performance of the binary classifier while PRC curve gives us a summary about the binary classifier's overall predictive performance. While the ROC curve is a plot of TPR against FPR, PRC is a plot of precision against recall.

1.2.7 Feature Ranking:

As we have 13 features in our dataset, all these 13 features do not influence the outcome (the target) in the same way. Some of these features might influence the outcome heavily while some might have very little influence on the outcome. By selecting only the most relevant features, we can improve the efficiency as well as the performance of our model. Feature Ranking was performed to find out the most relevant 5 features for all models except KNN and MLP. As KNN depends on the distance between the data points, one cannot remove any feature without knowing the proper distribution of data. As for MLP, they are already robust to less influential features.

CHAPTER 2

MERITS AND DEMERITS OF THE BASE PAPER

2.1 Literature Survey:

- In 2011, Jyoti Soni et al. implemented Naïve Bayes (NB), Decision Tree (DT) and Classification via clustering on a dataset containing 909 records and 13 attributes. Decision Tree gave the best result with an accuracy of 99.2%.
- In 2018, Mustafa Jan et al. implemented NB, ANN, RF, SVM, LR and ensemble approach with all above mentioned algorithms on Cleveland and Hungarian heart disease related dataset collected from the UCI repository. Random Forest produced the best accuracy with 98.136%.
- In 2018, Hung Minh Le et al. took 4 public heart disease datasets which are available for free in the UCI repository and implemented feature ranking. Then, they implemented NB, LR and SVM with linear and non-linear kernels. SVM with linear kernels outperformed other models and gave an accuracy of 89.93%.
- In 2019, Senthil Kumar Mohan et al. implemented NB, Generalized linear model, LR, Deep Learning, DT, RF, Gradient Boosted Types, SVM, VOTE and Hybrid Random Forest with Linear Model (HRFLM) on Cleveland dataset for heart disease from UCI repository. HRFLM gave the highest accuracy of 88.4%.
- In 2019, C. Beulah et al. took the Cleveland heart disease dataset and implemented Bayes Net (BN), NB, RF, C-4.5 (type of DT algorithm), Multilayer Perceptron (MLP), Projective Adaptive Resonance Theory (PART) along with the ensemble techniques. Majority Vote with BN, NB, MLP and RF gave an accuracy of 85.48% which was higher than the other model.

2.2 Merits of Base Paper:

- Using Supervised Machine Learning models, heart disease predictions can be made easily and in a very efficient manner.
- The use of Supervised Machine Learning models instead of Unsupervised Machine learning models means our models is set to be more accurate.
- A total of six supervised machine learning algorithm were implemented.
- KNN algorithm gives 100% accuracy which means it gives us reliable predictions without any error.

2.3 Demerits of Base Paper:

- Support Vector Machines another type of algorithm which comes under Supervised Learning algorithms was not implemented.
- The amount of data available in the dataset were comparatively less compared to the amount of real time healthcare data that exists.

2.4 Metrics of Different Models:

	ABM1	KNN	LR	DT	RF	MLP
Accuracy(%)	99.792	100.0	86.271	99.372	99.581	95.175
Recall	1.0	1.0	0.902	1.0	1.0	0.947
Precision	0.996	1.0	0.839	0.988	0.992	0.958
F1-Score	0.998	1.0	0.868	0.994	0.996	0.952
MCC-Score	0.996	1.0	0.726	0.988	0.992	0.902
Kappa-Score	0.996	1.0	0.72	0.987	0.991	0.899
Sensitivity	1.0	1.0	0.902	1.0	1.0	0.947
Specificity	0.996	1.0	0.82	0.987	0.992	0.945
AUROC	0.998	1.0	0.861	0.993	0.996	0.946
AUPRC	0.996	1.0	0.807	0.988	0.992	0.933

Table 2.1 – Metrics of Different Models

CHAPTER 3

SOURCE CODE

Importing Necessary Packages:

```
import numpy as np

import pandas as pd

import seaborn as sns

import matplotlib.pyplot as plt
```

Reading the CSV File:

```
heart_data = pd.read_csv("D:\Sastra\Sem 6\Mini Project\heart.csv")

heart_data
```

Basic Data Exploration:

```
heart_data.head() #Prints first 5 rows

heart_data.isnull().any() #To check null values in the dataset

heart_data.describe() #finding the description of the dataset

heart_data.shape #Returns the Size of the Dataset
```

Finding outliers using Boxplot and removing them based on IQR Score

```
#Finding Outliers in 'age' Attribute

sns.boxplot(x=heart_data["age"])

#Finding Outliers in 'sex' Attribute

sns.boxplot(x=heart_data["sex"])

#Finding Outliers in 'cp' Attribute

sns.boxplot(x=heart_data["cp"])

#Finding Outliers in 'trestbps' Attribute

sns.boxplot(x=heart_data["trestbps"])

#Finding Upper, Middle and lower Quartiles

Q1 = np.percentile(heart_data['trestbps'], 25, interpolation = 'midpoint')

Q2 = np.percentile(heart_data['trestbps'], 50, interpolation = 'midpoint')

Q3 = np.percentile(heart_data['trestbps'], 75, interpolation = 'midpoint')

#Finding IQR
```

```
IQR = Q3-Q1

#Finding b_lower and b_upper

b_lower = Q1 - 1.5*IQR

b_upper = Q3 + 1.5*IQR

#Removing Outliers

cleaned_heart_data = heart_data[(heart_data.trestbps>b_lower) & (heart_data.trestbps<b_upper)]

#Outlier-Free 'trestbps' Attribute

sns.boxplot(x=cleaned_heart_data["trestbps"])

#Finding Outliers in 'chol' Attribute

sns.boxplot(x=heart_data["chol"])

#Finding Upper, Middle and lower Quartiles

Q1 = np.percentile(heart_data['chol'], 25, interpolation = 'midpoint')

Q2 = np.percentile(heart_data['chol'], 50, interpolation = 'midpoint')

Q3 = np.percentile(heart_data['chol'], 75, interpolation = 'midpoint')

#Finding IQR

IQR = Q3-Q1

#Finding b_lower and b_upper

b_lower = Q1 - 1.5*IQR

b_upper = Q3 + 1.5*IQR

#Removing Outliers

cleaned_heart_data = cleaned_heart_data[(cleaned_heart_data.chol>b_lower) & (cleaned_heart_data.chol<b_upper)]

#Outlier-Free 'chol' Attribute

sns.boxplot(x=cleaned_heart_data["chol"])

#Finding Outliers in 'restecg' Attribute

sns.boxplot(x=heart_data["restecg"])

#Finding Outliers in 'thalach' Attribute

sns.boxplot(x=heart_data["thalach"])

#Finding Upper, Middle and lower Quartiles
```

```
Q1 = np.percentile(heart_data['thalach'], 25, interpolation = 'midpoint')

Q2 = np.percentile(heart_data['thalach'], 50, interpolation = 'midpoint')

Q3 = np.percentile(heart_data['thalach'], 75, interpolation = 'midpoint')

#Finding IQR

IQR = Q3-Q1

#Finding b_lower and b_upper

b_lower = Q1 - 1.5*IQR

b_upper = Q3 + 1.5*IQR

#Removing Outliers

cleaned_heart_data = cleaned_heart_data[(cleaned_heart_data.thalach>b_lower) & (cleaned_heart_data.thalach<b_upper)]

#Outlier-Free 'thalach' Attribute

sns.boxplot(x=cleaned_heart_data["thalach"])

#Finding Outliers in 'exang' Attribute

sns.boxplot(x=heart_data["exang"])

#Finding Outliers in 'ca' Attribute

sns.boxplot(x=heart_data["ca"])

#Finding Upper, Middle and lower Quartiles

Q1 = np.percentile(heart_data['ca'], 25, interpolation = 'midpoint')

Q2 = np.percentile(heart_data['ca'], 50, interpolation = 'midpoint')

Q3 = np.percentile(heart_data['ca'], 75, interpolation = 'midpoint')

#Finding IQR

IQR = Q3-Q1

#Finding b_lower and b_upper

b_lower = Q1 - (1.5*IQR)

b_upper = Q3 + (1.5*IQR)

#Removing Outliers

cleaned_heart_data = cleaned_heart_data[(cleaned_heart_data.ca>b_lower) & (cleaned_heart_data.ca<b_upper)]

#Outlier-Free 'ca' Attribute
```

```
sns.boxplot(x=cleaned_heart_data["ca"])

#Finding Outliers in 'slope' Attribute

sns.boxplot(x=heart_data["slope"])

#Finding Outliers in 'thal' Attribute

sns.boxplot(x=heart_data["thal"])

#Finding Upper, Middle and lower Quartiles

Q1 = np.percentile(heart_data['thal'], 25, interpolation = 'midpoint')

Q2 = np.percentile(heart_data['thal'], 50, interpolation = 'midpoint')

Q3 = np.percentile(heart_data['thal'], 75, interpolation = 'midpoint')

#Finding IQR

IQR = Q3-Q1

#Finding b_lower and b_upper

b_lower = Q1 - (1.5*IQR)

b_upper = Q3 + (1.5*IQR)

#Removing Outliers

cleaned_heart_data = cleaned_heart_data[(cleaned_heart_data.thal>b_lower) & (cleaned_heart_data.thal<b_upper)]

#Outlier-Free 'thal' Attribute

sns.boxplot(x=cleaned_heart_data["thal"])

#Finding Outliers in 'oldpeak' Attribute

sns.boxplot(x=heart_data["oldpeak"])

#Finding Upper, Middle and lower Quartiles

Q1 = np.percentile(heart_data['oldpeak'], 25, interpolation = 'midpoint')

Q2 = np.percentile(heart_data['oldpeak'], 50, interpolation = 'midpoint')

Q3 = np.percentile(heart_data['oldpeak'], 75, interpolation = 'midpoint')

#Finding IQR

IQR = Q3-Q1

#Finding b_lower and b_upper

b_lower = Q1 - (1.5*IQR)
```

```
b_upper = Q3 + (1.5*IQR)

#Removing Outliers

cleaned_heart_data = cleaned_heart_data[(cleaned_heart_data.oldpeak>b_lower) & (cleaned_heart_data.oldpeak<b_upper)]

#Outlier-Free 'oldpeak' Attribute

sns.boxplot(x=cleaned_heart_data["oldpeak"])

# count plot to check if the dataset is balanced

sns.countplot(cleaned_heart_data['target'])

# Show the plot

plt.show()
```

Applying SMOTE to Balance the Dataset

```
X = cleaned_heart_data.drop("target",axis=True)

y = cleaned_heart_data["target"]

print("Before OverSampling, counts of label '1': {}".format(sum(y == 1)))

print("Before OverSampling, counts of label '0': {} \n".format(sum(y == 0)))

# Import SMOTE module from imblearn library

from imblearn.over_sampling import SMOTE

#Applying SMOTE

sm = SMOTE(random_state = 2)

X, y = sm.fit_resample(X, y)

print('After OverSampling, the shape of X: {}'.format(X.shape))

print('After OverSampling, the shape of Y: {} \n'.format(y.shape))

print("After OverSampling, counts of label '1': {}".format(sum(y == 1)))

print("After OverSampling, counts of label '0': {}".format(sum(y == 0)))

print("\n\nAfter Applying SMOTE, we have balanced the Dataset (i.e) We have made the number of counts of Diseased and Non-Diseased Samples equal")

#Creating new DataFrame with new values after applying SMOTE

final_heart_data=X

# count plot to make sure the dataset is balanced

import seaborn as sns
```

```
import matplotlib.pyplot as plt

sns.countplot(final_heart_data['target'])

# Show the plot

plt.show()
```

Detecting Correlation among attributes:

```
#Applying heatmap to find the correlation

sns.set (rc = {'figure.figsize':(15, 8)})

sns.heatmap(final_heart_data.corr(),cmap="Blues",annot=True)
```

KDE Plot

```
# KDE plot for both diseased and non-diseased individuals according to age distribution.

sns.set (rc = {'figure.figsize':(10, 8)})

sns.kdeplot(data=final_heart_data, x=final_heart_data['age'],
hue=final_heart_data['target'], multiple='stack')

plt.show()
```

AdaBoostM1 Implementation:

```
# Splitting the dataset into X and Y

X=final_heart_data.drop("target",axis=1)

Y=final_heart_data["target"]

auc_roc_scores=[]

auc_prc_scores=[]

#Importing Necessary Packages Required

from sklearn.metrics import
confusion_matrix,accuracy_score,precision_score,recall_score,f1_score,cohen_kappa_sc
ore,matthews_corrcoef,roc_auc_score

from sklearn.model_selection import KFold

from sklearn.metrics import auc, precision_recall_curve,average_precision_score

from sklearn.ensemble import AdaBoostClassifier

#Applying GridSearchCV to find out the best parameters


from sklearn.model_selection import GridSearchCV
```

```
n_estimators = list(range(100,1000,100))
grid={"n_estimators":n_estimators,'learning_rate':[1.0]}
ab_classifier=AdaBoostClassifier() #Creating AdaBoostClassifer classifier
ab_classifier_cv=GridSearchCV(ab_classifier,grid,cv=10)
ab_classifier_cv.fit(X,Y) # fitting the model for grid search
print("tuned hpyerparameters :(best parameters) ",ab_classifier_cv.best_params_)
#The parameters are set based on the value obtained from GridSearchCV method
ab_classifier=AdaBoostClassifier(n_estimators=700,learning_rate=1.0)
#Applying 10-fold cross validation
k = 10 #Setting k value as 10
k_fold = KFold(n_splits = k, random_state = None,shuffle=True)
#Initializing lists to store the values of each fold
acc_scores = []
rs_scores = []
ps_scores = []
sensitivity_scores = []
specificity_scores=[]
kappa_scores=[]
mcc_scores=[]
f1_scores=[]
auroc_scores=[]
auprc_scores=[]
# Looping over each split to get the accuracy score of each split
for training_index, testing_index in k_fold.split(X):
    X_train, X_test = X.iloc[training_index,:], X.iloc[testing_index,:]
    Y_train, Y_test = Y.iloc[training_index] , Y.iloc[testing_index]
    # Fitting training data to the model
    ab_classifier.fit(X_train,Y_train)
    # Predicting values for the testing dataset
```

```
Y_pred = ab_classifier.predict(X_test)
    #Creating Confusion Matrix to find the TP,TN,FP,FN values
    conf_matrix=confusion_matrix(Y_test,Y_pred)
    # Calculating accuracy and other metrics
    acc = accuracy_score(Y_test , Y_pred)
    rs = recall_score(Y_test,Y_pred)
    pc = precision_score(Y_test,Y_pred)
    f1s = f1_score(Y_test,Y_pred)
    cks = cohen_kappa_score(Y_test,Y_pred)
    mcc = matthews_corrcoef(Y_test,Y_pred)
    auroc = roc_auc_score(Y_test, Y_pred)
    auprc = average_precision_score(Y_test, Y_pred)
    #Extracting TP,TN,FP,FN values from the confusion matrix
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
    #Calculating sensitivity and speceficity
    sensitivity = (TP / float(TP + FN))
    specificity = (TN / float(TN + FP))
    #Appending the values of each fold to the list created earlier
    acc_scores.append(acc)
    rs_scores.append(rs)
    ps_scores.append(pc)
    f1_scores.append(f1s)
    auroc_scores.append(auroc)
    auprc_scores.append(auprc)
    mcc_scores.append(mcc)
    kappa_scores.append(cks)
```

```
    sensitivity_scores.append(sensitivity)
    specificity_scores.append(specificity)
# Calculating mean value of all the metrics from their respective lists
mean_acc_score = sum(acc_scores) / k
mean_rs_score = sum(rs_scores) / k
mean_ps_score = sum(ps_scores) / k
mean_f1_score = sum(f1_scores) / k
mean_mcc_score = sum(mcc_scores) / k
mean_auroc_score = sum(auroc_scores) / k
mean_auprc_score = sum(auprc_scores) / k
mean_kappa_score = sum(kappa_scores) / k
mean_sensitivity_score = sum(sensitivity_scores) / k
mean_specificity_score = sum(specificity_scores) / k
#Rounding off the values
mean_acc_score = (round(mean_acc_score, 5))*100
mean_rs_score = round(mean_rs_score, 3)
mean_ps_score = round(mean_ps_score, 3)
mean_f1_score = round(mean_f1_score, 3)
mean_mcc_score = round(mean_mcc_score, 3)
mean_kappa_score = round(mean_kappa_score, 3)
mean_sensitivity_score = round(mean_sensitivity_score, 3)
mean_specificity_score = round(mean_specificity_score, 3)
mean_auroc_score = round(mean_auroc_score, 3)
mean_auprc_score = round(mean_auprc_score, 3)
#Printing all the metrics
print("Mean accuracy score: ", mean_acc_score)
print("Mean recall score: ", mean_rs_score)
print("Mean Precison score: ", mean_ps_score)
print("Mean F1-score score: ", mean_f1_score)
```

print("Mean MCC score: ", mean_mcc_score)

print("Mean kappa score: ", mean_kappa_score)

print("Mean sensitivity score: ", mean_sensitivity_score)

print("Mean speceficity score: ", mean_specificity_score)

print("Mean AUROC value: ", mean_auroc_score)

print("Mean AUPRC value: ", mean_auprc_score)

auc_roc_scores.append(mean_auroc_score)

auc_prc_scores.append(mean_auprc_score)

from prettytable import PrettyTable

Specify the Column Names while initializing the Table

metric_table = PrettyTable(["Model Name", "Accuracy", "Recall", "Precision",'F1 score','MCC-Score','kappa-score','sensitivity','specificity','AUROC','AUPRC'])

Add rows

metric_table.add_row(["AdaBoostM1",mean_acc_score,mean_rs_score,mean_ps_score, mean_f1_score,mean_mcc_score,mean_kappa_score,mean_sensitivity_score,mean_specificity_score,mean_auroc_score,mean_auprc_score])

print(metric_table)

K-Nearest Neighbor Implementation:

Splitting the dataset into X and Y

X=final_heart_data.drop("target",axis=1)

Y=final_heart_data["target"]

#Importing Necessary Packages Required

from sklearn.metrics import

confusion_matrix,accuracy_score,precision_score,recall_score,f1_score,cohen_kappa_score,matthews_corrcoef,roc_auc_score

from sklearn.model_selection import KFold

from sklearn.metrics import auc, precision_recall_curve,average_precision_score

from sklearn.neighbors import KNeighborsClassifier

#Applying GridSearchCV to find out the best parameters

from sklearn.model_selection import GridSearchCV

knn_classifier = KNeighborsClassifier()

```
k_range = list(range(1, 31))

param_grid = [

    {'n_neighbors':k_range,

    'metric':['minkowski'],

     'algorithm':['auto', 'ball_tree', 'kd_tree', 'brute'],

    'p':[1,2]}

]

# defining parameter range

knn_grid = GridSearchCV(knn_classifier, param_grid, cv=10, scoring='accuracy',
return_train_score=False,verbose=1)

# fitting the model for grid search

knn_grid_search=knn_grid.fit(X,Y)

print("tuned hpyerparameters :(best parameters) ",knn_grid_search.best_params_)

#The parameters are set based on the value obtained from GridSearchCV method

knn_classifier =
KNeighborsClassifier(n_neighbors=1,metric='minkowski',p=1,algorithm='auto')

#Applying 10-fold cross validation

k = 10 #Setting k value as 10

k_fold = KFold(n_splits = k, random_state = None,shuffle=True)

#Initializing lists to store the values of each fold

acc_scores = []

rs_scores = []

ps_scores = []

sensitivity_scores = []

specificity_scores=[]

kappa_scores=[]

mcc_scores=[]

f1_scores=[]

auroc_scores=[]

auprc_scores=[]
```

```
# Looping over each split to get the accuracy score of each split
for training_index, testing_index in k_fold.split(X):
    X_train, X_test = X.iloc[training_index,:], X.iloc[testing_index,:]
    Y_train, Y_test = Y.iloc[training_index] , Y.iloc[testing_index]
# Fitting training data to the model
    knn_classifier.fit(X_train,Y_train)
    # Predicting values for the testing dataset
    Y_pred = knn_classifier.predict(X_test)
 #Creating Confusion Matrix to find the TP,TN,FP,FN values
    conf_matrix=confusion_matrix(Y_test,Y_pred)
    # Calculating accuracy and other metrics
    acc = accuracy_score(Y_test , Y_pred)
    rs = recall_score(Y_test,Y_pred)
    pc = precision_score(Y_test,Y_pred)
    f1s = f1_score(Y_test,Y_pred)
    cks = cohen_kappa_score(Y_test,Y_pred)
    mcc = matthews_corrcoef(Y_test,Y_pred)
    auroc = roc_auc_score(Y_test, Y_pred)
    auprc = average_precision_score(Y_test, Y_pred)
#Extracting TP,TN,FP,FN values from the confusion matrix
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
  #Calculating sensitivity and speceficity
    sensitivity = (TP / float(TP + FN))
    specificity = (TN / float(TN + FP))
  #Appending the values of each fold to the list created earlier
    acc_scores.append(acc)
```

```
rs_scores.append(rs)
    ps_scores.append(pc)
    f1_scores.append(f1s)
    auroc_scores.append(auroc)
    auprc_scores.append(auprc)
    mcc_scores.append(mcc)
    kappa_scores.append(cks)
    sensitivity_scores.append(sensitivity)
    specificity_scores.append(specificity)
# Calculating mean value of all the metrics from their respective lists
mean_acc_score = sum(acc_scores) / k
mean_rs_score = sum(rs_scores) / k
mean_ps_score = sum(ps_scores) / k
mean_f1_score = sum(f1_scores) / k
mean_mcc_score = sum(mcc_scores) / k
mean_auroc_score = sum(auroc_scores) / k
mean_auprc_score = sum(auprc_scores) / k
mean_kappa_score = sum(kappa_scores) / k
mean_sensitivity_score = sum(sensitivity_scores) / k
mean_specificity_score = sum(specificity_scores) / k
#Rounding off the values
mean_acc_score = (round(mean_acc_score, 5))*100
mean_rs_score = round(mean_rs_score, 3)
mean_ps_score = round(mean_ps_score, 3)
mean_f1_score = round(mean_f1_score, 3)
mean_mcc_score = round(mean_mcc_score, 3)
mean_kappa_score = round(mean_kappa_score, 3)
mean_sensitivity_score = round(mean_sensitivity_score, 3)
mean_specificity_score = round(mean_specificity_score, 3)
```

```
mean_auroc_score = round(mean_auroc_score, 3)
mean_auprc_score = round(mean_auprc_score, 3)
#Printing all the metrics
print("Mean accuracy score: ", mean_acc_score)
print("Mean recall score: ", mean_rs_score)
print("Mean Precison score: ", mean_ps_score)
print("Mean F1-score score: ", mean_f1_score)
print("Mean MCC score: ", mean_mcc_score)
print("Mean kappa score: ", mean_kappa_score)
print("Mean sensitivity score: ", mean_sensitivity_score)
print("Mean specificity score: ", mean_specificity_score)
print("Mean AUROC value: ", mean_auroc_score)
print("Mean AUPRC value: ", mean_auprc_score)
auc_roc_scores.append(mean_auroc_score)
auc_prc_scores.append(mean_auprc_score)
from prettytable import PrettyTable
# Specify the Column Names while initializing the Table
metric_table = PrettyTable(["Model Name", "Accuracy", "Recall", "Precision",'F1-score','MCC-Score','kappa-score','sensitivity','specificity','AUROC','AUPRC'])
# Add rows
metric_table.add_row(["KNN", mean_acc_score, mean_rs_score, mean_ps_score, mean_f1_score, mean_mcc_score, mean_kappa_score, mean_sensitivity_score, mean_specificity_score, mean_auroc_score,mean_auprc_score])
print(metric_table)
```

Logistic Regression Implementation:

```
# Splitting the dataset into X and Y
X=final_heart_data.drop("target",axis=1)
Y=final_heart_data["target"]
#Importing Necessary Packages Required
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score, cohen_kappa_score,matthews_corrcoef,roc_auc_score
```

```
from sklearn.model_selection import KFold
from sklearn.metrics import auc, precision_recall_curve,average_precision_score
from sklearn.linear_model import LogisticRegression
#Applying GridSearchCV to find out the best parameters
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
grid={"penalty":["l1","l2"],"max_iter":[100,200,300,400,500],"solver":['newton-cg',
'lbfgs', 'liblinear', 'saag', 'saga']}
lr_classifier=LogisticRegression()
lr_grid=GridSearchCV(lr_classifier,grid,cv=10)
lr_grid_search=lr_grid.fit(X,Y)
print("tuned hpyerparameters :(best parameters) ",lr_grid_search.best_params_)
#The parameters are set based on the value obtained from GridSearchCV method
lr_classifier=LogisticRegression(C=1.0, penalty = 'l2',max_iter=400,solver='lbfgs')
#Applying 10-fold cross validation
k = 10 #Setting k value as 10
k_fold = KFold(n_splits = k, random_state = None,shuffle=True)
#Initializing lists to store the values of each fold
acc_scores = []
rs_scores = []
ps_scores = []
sensitivity_scores = []
specificity_scores=[]
kappa_scores=[]
mcc_scores=[]
f1_scores=[]
auroc_scores=[]
auprc_scores=[]
# Looping over each split to get the accuracy score of each split
```

```
for training_index, testing_index in k_fold.split(X):
    X_train, X_test = X.iloc[training_index,:], X.iloc[testing_index,:]
    Y_train, Y_test = Y.iloc[training_index] , Y.iloc[testing_index]
  # Fitting training data to the model
    lr_classifier.fit(X_train,Y_train)
   # Predicting values for the testing dataset
Y_pred = lr_classifier.predict(X_test)
   #Creating Confusion Matrix to find the TP,TN,FP,FN values
    conf_matrix=confusion_matrix(Y_test,Y_pred)
   # Calculating accuracy and other metrics
    acc = accuracy_score(Y_test , Y_pred)
    rs = recall_score(Y_test,Y_pred)
    pc = precision_score(Y_test,Y_pred)
    f1s = f1_score(Y_test,Y_pred)
    cks = cohen_kappa_score(Y_test,Y_pred)
    mcc = matthews_corrcoef(Y_test,Y_pred)
    auroc = roc_auc_score(Y_test, Y_pred)
    auprc = average_precision_score(Y_test, Y_pred)
    #Extracting TP,TN,FP,FN values from the confusion matrix
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
   #Calculating sensitivity and speceficity
    sensitivity = (TP / float(TP + FN))
    specificity = (TN / float(TN + FP))
   #Appending the values of each fold to the list created earlier
    acc_scores.append(acc)
    rs_scores.append(rs)
```

```
    ps_scores.append(pc)
    f1_scores.append(f1s)
    auroc_scores.append(auroc)
    auprc_scores.append(auprc)
    mcc_scores.append(mcc)
    kappa_scores.append(cks)
sensitivity_scores.append(sensitivity)
    specificity_scores.append(specificity)
 # Calculating mean value of all the metrics from their respective lists
mean_acc_score = sum(acc_scores) / k
mean_rs_score = sum(rs_scores) / k
mean_ps_score = sum(ps_scores) / k
mean_f1_score = sum(f1_scores) / k
mean_mcc_score = sum(mcc_scores) / k
mean_auroc_score = sum(auroc_scores) / k
mean_auprc_score = sum(auprc_scores) / k
mean_kappa_score = sum(kappa_scores) / k
mean_sensitivity_score = sum(sensitivity_scores) / k
mean_specificity_score = sum(specificity_scores) / k
#Rounding off the values
mean_acc_score = (round(mean_acc_score, 5))*100
mean_rs_score = round(mean_rs_score, 3)
mean_ps_score = round(mean_ps_score, 3)
mean_f1_score = round(mean_f1_score, 3)
mean_mcc_score = round(mean_mcc_score, 3)
mean_kappa_score = round(mean_kappa_score, 3)
mean_sensitivity_score = round(mean_sensitivity_score, 3)
mean_specificity_score = round(mean_specificity_score, 3)
mean_auroc_score = round(mean_auroc_score, 3)
```

```
mean_auprc_score = round(mean_auprc_score, 3)

#Printing all the metrics

print("Mean accuracy score: ", mean_acc_score)

print("Mean recall score: ", mean_rs_score)

print("Mean Precison score: ", mean_ps_score)

print("Mean F1-score score: ", mean_f1_score)

print("Mean MCC score: ", mean_mcc_score)

print("Mean kappa score: ", mean_kappa_score)

print("Mean sensitivity score: ", mean_sensitivity_score)

print("Mean specificity score: ", mean_specificity_score)

print("Mean AUROC value: ", mean_auroc_score)

print("Mean AUPRC value: ", mean_auprc_score)

auc_roc_scores.append(mean_auroc_score)

auc_prc_scores.append(mean_auprc_score)

from prettytable import PrettyTable

#Specify the Column Names while initializing the Table

metric_table = PrettyTable(["Model Name", "Accuracy", "Recall", "Precision",'F1-score','MCC-Score','kappa-score','sensitivity','specificity','AUROC','AUPRC'])

# Add rows

metric_table.add_row(["LR", mean_acc_score, mean_rs_score, mean_ps_score, mean_f1_score, mean_mcc_score, mean_kappa_score, mean_sensitivity_score, mean_specificity_score,  mean_auroc_score,mean_auprc_score])

print(metric_table)
```

Decision Tree Implementation:

```
# Splitting the dataset into X and Y

X=final_heart_data.drop("target",axis=1)

Y=final_heart_data["target"]

#Importing Necessary Packages Required

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score,cohen_kappa_score,matthews_corrcoef,roc_auc_score

from sklearn.model_selection import KFold
```

```
from sklearn.metrics import auc, precision_recall_curve,average_precision_score

from sklearn.tree import DecisionTreeClassifier

#Applying GridSearchCV to find out the best parameters

from sklearn.model_selection import GridSearchCV


dt_classifier = DecisionTreeClassifier()

maxd_range = list(range(1, 31))

min_sample_split=list(range(1,10))

param_grid = [

    {'max_depth': maxd_range,

    'criterion': ["gini", "entropy"],

    'min_samples_split':min_sample_split,

    'max_features': ['auto', 'sqrt', 'log2']}

]

# defining parameter range

dt_grid = GridSearchCV(dt_classifier, param_grid, cv=10, scoring='accuracy',
return_train_score=False,verbose=1)

# fitting the model for grid search

dt_grid_search=dt_grid.fit(X,Y)

print("tuned hpyerparameters :(best parameters) ",dt_grid_search.best_params_)

#The parameters are set based on the value obtained from GridSearchCV method

dt_classifier = DecisionTreeClassifier(criterion= 'gini', max_depth=
20,max_features='sqrt',min_samples_split=2)

#Applying 10-fold cross validation

k = 10 #Setting k value as 10

k_fold = KFold(n_splits = k, random_state = None,shuffle=True)

#Initializing lists to store the values of each fold

acc_scores = []

rs_scores = []

ps_scores = []
```

```
sensitivity_scores = []
specificity_scores=[]
kappa_scores=[]
mcc_scores=[]
f1_scores=[]
auroc_scores=[]
auprc_scores=[]
# Looping over each split to get the accuracy score of each split
for training_index, testing_index in k_fold.split(X):
    X_train, X_test = X.iloc[training_index,:], X.iloc[testing_index,:]
    Y_train, Y_test = Y.iloc[training_index] , Y.iloc[testing_index]
   # Fitting training data to the model
    dt_classifier.fit(X_train,Y_train)
   # Predicting values for the testing dataset
    Y_pred = dt_classifier.predict(X_test)
    #Creating Confusion Matrix to find the TP,TN,FP,FN values
      conf_matrix=confusion_matrix(Y_test,Y_pred)
    # Calculating accuracy and other metrics
    acc = accuracy_score(Y_test , Y_pred)
    rs = recall_score(Y_test,Y_pred)
    pc = precision_score(Y_test,Y_pred)
    f1s = f1_score(Y_test,Y_pred)
    cks = cohen_kappa_score(Y_test,Y_pred)
    mcc = matthews_corrcoef(Y_test,Y_pred)
    auroc = roc_auc_score(Y_test, Y_pred)
    auprc = average_precision_score(Y_test, Y_pred)
#Extracting TP,TN,FP,FN values from the confusion matrix
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
```

```
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
    #Calculating sensitivity and speceficity
    sensitivity = (TP / float(TP + FN))
    specificity = (TN / float(TN + FP))
#Appending the values of each fold to the list created earlier
    acc_scores.append(acc)
    rs_scores.append(rs)
    ps_scores.append(pc)
    f1_scores.append(f1s)
    auroc_scores.append(auroc)
    auprc_scores.append(auprc)
    mcc_scores.append(mcc)
    kappa_scores.append(cks)
    sensitivity_scores.append(sensitivity)
    specificity_scores.append(specificity)
# Calculating mean value of all the metrics from their respective lists
mean_acc_score = sum(acc_scores) / k
mean_rs_score = sum(rs_scores) / k
mean_ps_score = sum(ps_scores) / k
mean_f1_score = sum(f1_scores) / k
mean_mcc_score = sum(mcc_scores) / k
mean_auroc_score = sum(auroc_scores) / k
mean_auprc_score = sum(auprc_scores) / k
mean_kappa_score = sum(kappa_scores) / k
mean_sensitivity_score = sum(sensitivity_scores) / k
mean_specificity_score = sum(specificity_scores) / k
#Rounding off the values
mean_acc_score = (round(mean_acc_score, 5))*100
```

```
mean_rs_score = round(mean_rs_score, 3)
mean_ps_score = round(mean_ps_score, 3)
mean_f1_score = round(mean_f1_score, 3)
mean_mcc_score = round(mean_mcc_score, 3)
mean_kappa_score = round(mean_kappa_score, 3)
mean_sensitivity_score = round(mean_sensitivity_score, 3)
mean_specificity_score = round(mean_specificity_score, 3)
mean_auroc_score = round(mean_auroc_score, 3)
mean_auprc_score = round(mean_auprc_score, 3)
#Printing all the metrics
print("Mean accuracy score: ", mean_acc_score)
print("Mean recall score: ", mean_rs_score)
print("Mean Precison score: ", mean_ps_score)
print("Mean F1-score score: ", mean_f1_score)
print("Mean MCC score: ", mean_mcc_score)
print("Mean kappa score: ", mean_kappa_score)
print("Mean sensitivity score: ", mean_sensitivity_score)
print("Mean specificity score: ", mean_specificity_score)
print("Mean AUROC value: ", mean_auroc_score)
print("Mean AUPRC value: ", mean_auprc_score)
auc_roc_scores.append(mean_auroc_score)
auc_prc_scores.append(mean_auprc_score)
from prettytable import PrettyTable
#Specify the Column Names while initializing the Table
metric_table = PrettyTable(["Model Name", "Accuracy", "Recall", "Precision",'F1-score','MCC-Score','kappa-score','sensitivity','specificity','AUROC','AUPRC'])
# Add rows
metric_table.add_row(["DT",round(mean_acc_score,3), mean_rs_score, mean_ps_score, mean_f1_score,mean_mcc_score,mean_kappa_score,mean_sensitivity_score,mean_specificity_score, mean_auroc_score,mean_auprc_score])
```

print(metric_table)

Random Forest Implementation:

#Importing Necessary Packages Required

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,f1_score,cohen_kappa_score,matthews_corrcoef,roc_auc_score

from sklearn.model_selection import KFold

from sklearn.metrics import auc, precision_recall_curve,average_precision_score

from sklearn.ensemble import RandomForestClassifier

#Applying GridSearchCV to find out the best parameters

from sklearn.model_selection import GridSearchCV

rf_classifier = RandomForestClassifier()

n_estimator= list(range(100,1000,100))

param_grid = [

```
    {
        'n_estimators':n_estimator,
        'max_features': ['sqrt'],
       'criterion':['gini'],
       'max_depth':[20],
       'min_samples_split':[2]
    }
]
```

defining parameter range

rf_grid = GridSearchCV(rf_classifier, param_grid, cv=10, scoring='accuracy', return_train_score=False,verbose=1)

fitting the model for grid search

rf_grid_search=rf_grid.fit(X,Y)

print("tuned hpyerparameters :(best parameters) ",rf_grid_search.best_params_)

#The parameters are set based on the value obtained from GridSearchCV method

rf_classifier = RandomForestClassifier(n_estimators=300,min_samples_split=2,max_features='sqrt',criterion= 'gini', max_depth= 20)

```
#Applying 10-fold cross validation
k = 10 #Setting k value as 10
k_fold = KFold(n_splits = k, random_state = None,shuffle=True)
#Initializing lists to store the values of each fold
acc_scores = []
rs_scores = []
ps_scores = []
sensitivity_scores = []
specificity_scores=[]
kappa_scores=[]
mcc_scores=[]
f1_scores=[]
auroc_scores=[]
auprc_scores=[]
# Looping over each split to get the accuracy score of each split
for training_index, testing_index in k_fold.split(X):
    X_train, X_test = X.iloc[training_index,:], X.iloc[testing_index,:]
    Y_train, Y_test = Y.iloc[training_index] , Y.iloc[testing_index]
    # Fitting training data to the model
    rf_classifier.fit(X_train,Y_train)
    # Predicting values for the testing dataset
    Y_pred = rf_classifier.predict(X_test)
    #Creating Confusion Matrix to find the TP,TN,FP,FN values
      conf_matrix=confusion_matrix(Y_test,Y_pred)
    # Calculating accuracy and other metrics
    acc = accuracy_score(Y_test , Y_pred)
    rs = recall_score(Y_test,Y_pred)
    pc = precision_score(Y_test,Y_pred)
    f1s = f1_score(Y_test,Y_pred)
```

```
cks = cohen_kappa_score(Y_test,Y_pred)
   mcc = matthews_corrcoef(Y_test,Y_pred)
   auroc = roc_auc_score(Y_test, Y_pred)
   auprc = average_precision_score(Y_test, Y_pred)
  #Extracting TP,TN,FP,FN values from the confusion matrix
   TP = conf_matrix[1][1]
   TN = conf_matrix[0][0]
   FP = conf_matrix[0][1]
   FN = conf_matrix[1][0]
   #Calculating sensitivity and speceficity
   sensitivity = (TP / float(TP + FN))
   specificity = (TN / float(TN + FP))
   #Appending the values of each fold to the list created earlier
   acc_scores.append(acc)
   rs_scores.append(rs)
   ps_scores.append(pc)
   f1_scores.append(f1s)
   auroc_scores.append(auroc)
   auprc_scores.append(auprc)
   mcc_scores.append(mcc)
   kappa_scores.append(cks)
   sensitivity_scores.append(sensitivity)
   specificity_scores.append(specificity)
# Calculating mean value of all the metrics from their respective lists
mean_acc_score = sum(acc_scores) / k
mean_rs_score = sum(rs_scores) / k
mean_ps_score = sum(ps_scores) / k
mean_f1_score = sum(f1_scores) / k
mean_mcc_score = sum(mcc_scores) / k
```

```
mean_auroc_score = sum(auroc_scores) / k
mean_auprc_score = sum(auprc_scores) / k
mean_kappa_score = sum(kappa_scores) / k
mean_sensitivity_score = sum(sensitivity_scores) / k
mean_specificity_score = sum(specificity_scores) / k
#Rounding off the values
mean_acc_score = (round(mean_acc_score, 5))*100
mean_rs_score = round(mean_rs_score, 3)
mean_ps_score = round(mean_ps_score, 3)
mean_f1_score = round(mean_f1_score, 3)
mean_mcc_score = round(mean_mcc_score, 3)
mean_kappa_score = round(mean_kappa_score, 3)
mean_sensitivity_score = round(mean_sensitivity_score, 3)
mean_specificity_score = round(mean_specificity_score, 3)
mean_auroc_score = round(mean_auroc_score, 3)
mean_auprc_score = round(mean_auprc_score, 3)
#Printing all the metrics
print("Mean accuracy score: ", mean_acc_score)
print("Mean recall score: ", mean_rs_score)
print("Mean Precision score: ", mean_ps_score)
print("Mean F1-score score: ", mean_f1_score)
print("Mean MCC score: ", mean_mcc_score)
print("Mean kappa score: ", mean_kappa_score)
print("Mean sensitivity score: ", mean_sensitivity_score)
print("Mean specificity score: ", mean_specificity_score)
print("Mean AUROC value: ", mean_auroc_score)
print("Mean AUPRC value: ", mean_auprc_score)
auc_roc_scores.append(mean_auroc_score)
auc_prc_scores.append(mean_auprc_score)
```

from prettytable import PrettyTable

#Specify the Column Names while initializing the Table

metric_table = PrettyTable(["Model Name", "Accuracy", "Recall", "Precision",'F1-score','MCC-Score','kappa-score','sensitivity','specificity','AUROC','AUPRC'])

Add rows

metric_table.add_row(["RF", round(mean_acc_score,3), mean_rs_score, mean_ps_score, mean_f1_score,mean_mcc_score,mean_kappa_score,mean_sensitivity_score,mean_specificity_score,mean_auroc_score,mean_auprc_score])

print(metric_table)

Multi Layer Perceptron Implementation:

#Importing Necessary Packages Required

from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score,f1_score,cohen_kappa_score,matthews_corrcoef,roc_auc_score

from sklearn.model_selection import KFold

from sklearn.metrics import auc, precision_recall_curve,average_precision_score

from sklearn.neural_network import MLPClassifier

#The parameters are set based on the value obtained from GridSearchCV method

mlp_classifier = MLPClassifier()

#Applying GridSearchCV to find out the best parameters

from sklearn.model_selection import GridSearchCV

```
param_grid = [
    {
    'hidden_layer_sizes': [(100,100),(100,150),(150,100),(150,150),(200,150),(150,200),(200,200)],
    'activation': ['logistic', 'tanh', 'relu'],
    'solver': ['sgd', 'adam'],
    'alpha': [0.0001],
    'learning_rate': ['constant','adaptive']}
]
```

defining parameter range

```
mlp_grid = GridSearchCV(mlp_classifier, param_grid, cv=10, scoring='accuracy',
return_train_score=False,verbose=1)

# fitting the model for grid search

mlp_grid_search=mlp_grid.fit(X,Y)

print("tuned hpyerparameters :(best parameters) ",mlp_grid_search.best_params_)

print("tuned hpyerparameters :(best parameters) ",mlp_grid_search.best_score_)

#The parameters are set based on the value obtained from GridSearchCV method

mlp_classifier = MLPClassifier(max_iter=800,activation= 'tanh', alpha= 0.0001,
hidden_layer_sizes= (150, 200), learning_rate= 'constant', solver= 'adam')

#Applying 10-fold cross validation

k = 10 #Setting k value as 10

k_fold = KFold(n_splits = k, random_state = None,shuffle=True)

#Initializing lists to store the values of each fold

acc_scores = []

rs_scores = []

ps_scores = []

sensitivity_scores = []

specificity_scores=[]

kappa_scores=[]

mcc_scores=[]

f1_scores=[]

auroc_scores=[]

auprc_scores=[]

# Looping over each split to get the accuracy score of each split

for training_index, testing_index in k_fold.split(X):

    X_train, X_test = X.iloc[training_index,:], X.iloc[testing_index,:]

    Y_train, Y_test = Y.iloc[training_index] , Y.iloc[testing_index]

    # Fitting training data to the model

    mlp_classifier.fit(X_train,Y_train)

    # Predicting values for the testing dataset
```

```
    Y_pred = mlp_classifier.predict(X_test)
#Creating Confusion Matrix to find the TP,TN,FP,FN values
     conf_matrix=confusion_matrix(Y_test,Y_pred)
# Calculating accuracy and other metrics
    acc = accuracy_score(Y_test , Y_pred)
    rs = recall_score(Y_test,Y_pred)
    pc = precision_score(Y_test,Y_pred)
    f1s = f1_score(Y_test,Y_pred)
    cks = cohen_kappa_score(Y_test,Y_pred)
    mcc = matthews_corrcoef(Y_test,Y_pred)
    auroc = roc_auc_score(Y_test, Y_pred)
    auprc = average_precision_score(Y_test, Y_pred)
#Extracting TP,TN,FP,FN values from the confusion matrix
    TP = conf_matrix[1][1]
    TN = conf_matrix[0][0]
    FP = conf_matrix[0][1]
    FN = conf_matrix[1][0]
   #Calculating sensitivity and speceficity
    sensitivity = (TP / float(TP + FN))
    specificity = (TN / float(TN + FP))
   #Appending the values of each fold to the list created earlier
    acc_scores.append(acc)
    rs_scores.append(rs)
    ps_scores.append(pc)
    f1_scores.append(f1s)
    auroc_scores.append(auroc)
    auprc_scores.append(auprc)
    mcc_scores.append(mcc)
    kappa_scores.append(cks)
```

```
    sensitivity_scores.append(sensitivity)
    specificity_scores.append(specificity)
 # Calculating mean value of all the metrics from their respective lists
mean_acc_score = sum(acc_scores) / k
mean_rs_score = sum(rs_scores) / k
mean_ps_score = sum(ps_scores) / k
mean_f1_score = sum(f1_scores) / k
mean_mcc_score = sum(mcc_scores) / k
mean_auroc_score = sum(auroc_scores) / k
mean_auprc_score = sum(auprc_scores) / k
mean_kappa_score = sum(kappa_scores) / k
mean_sensitivity_score = sum(sensitivity_scores) / k
mean_specificity_score = sum(specificity_scores) / k
#Rounding off the values
mean_acc_score = (round(mean_acc_score, 5))*100
mean_rs_score = round(mean_rs_score, 3)
mean_ps_score = round(mean_ps_score, 3)
mean_f1_score = round(mean_f1_score, 3)
mean_mcc_score = round(mean_mcc_score, 3)
mean_kappa_score = round(mean_kappa_score, 3)
mean_sensitivity_score = round(mean_sensitivity_score, 3)
mean_specificity_score = round(mean_specificity_score, 3)
mean_auroc_score = round(mean_auroc_score, 3)
mean_auprc_score = round(mean_auprc_score, 3)
#Printing all the metrics
print("Mean accuracy score: ", mean_acc_score)
print("Mean recall score: ", mean_rs_score)
print("Mean Precision score: ", mean_ps_score)
print("Mean F1-score score: ", mean_f1_score)
```

```
print("Mean MCC score: ", mean_mcc_score)

print("Mean kappa score: ", mean_kappa_score)

print("Mean sensitivity score: ", mean_sensitivity_score)

print("Mean specificity score: ", mean_specificity_score)

print("Mean AUROC value: ", mean_auroc_score)

auc_roc_scores.append(mean_auroc_score)

auc_prc_scores.append(mean_auprc_score)

from prettytable import PrettyTable

#Specify the Column Names while initializing the Table

metric_table = PrettyTable(["Model Name", "Accuracy", "Recall", "Precision",'F1-score','MCC-Score','kappa-score','sensitivity','specificity','AUROC','AUPRC'])

 # Add rows

metric_table.add_row(["MLP", round(mean_acc_score,3), mean_rs_score, mean_ps_score, mean_f1_score, mean_mcc_score, mean_kappa_score, mean_sensitivity_score, mean_specificity_score,mean_auroc_score,mean_auprc_score])

print(metric_table)
```

AUROC Plot:

```
from sklearn.model_selection import train_test_split

X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.2)

from sklearn.metrics import roc_curve, roc_auc_score

# Instantiate the classfiers and make a list

classifiers = [lr_classifier,mlp_classifier,ab_classifier,knn_classifier,dt_classifier,rf_classifier,]

# Define a result table as a DataFrame

result_table = pd.DataFrame(columns=['classifiers', 'fpr','tpr','auc'])

auc_values = [auc_roc_scores[2], auc_roc_scores[5], auc_roc_scores[0], auc_roc_scores[1], auc_roc_scores[3],auc_roc_scores[4]]

# Train the models and record the results

i=0

for cls in classifiers:

    model = cls.fit(X_train, y_train)
```

```
    ypred = model.predict_proba(X_test)[::,1]
    fpr, tpr, _ = roc_curve(y_test,  yped)
    auc = auc_values[i]
    result_table = result_table.append({'classifiers':cls.__class__.__name__,
                                        'fpr':fpr,
                                        'tpr':tpr,
                                        'auc':auc}, ignore_index=True)
    i=i+1
# Set name of the classifiers as index labels
result_table.set_index('classifiers', inplace=True)
fig = plt.figure(figsize=(8,6))
for i in result_table.index:
    plt.plot(result_table.loc[i]['fpr'],
             result_table.loc[i]['tpr'],
             label="{}, AUC={:.3f}".format(i, result_table.loc[i]['auc']))
    plt.plot([0,1], [0,1], color='orange', linestyle='--')
plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("False Positive Rate", fontsize=15)
plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=15)
plt.title('ROC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(loc='best',fontsize=8)
plt.show()
```

AUPRC Plot:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(X,Y,test_size=0.2)
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
```

```
from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import auc,roc_curve, roc_auc_score,precision_recall_curve

# Instantiate the classfiers and make a list

classifiers = [lr_classifier, mlp_classifier, ab_classifier, knn_classifier, dt_classifier,
rf_classifier]

# Define a result table as a DataFrame

result_table1 = pd.DataFrame(columns=['classifiers', 'precision','recall','prc'])

prc_values = [auc_prc_scores[2], auc_prc_scores[5], auc_prc_scores[0],
auc_prc_scores[1], auc_prc_scores[3], auc_prc_scores[4]]

i=0

# Train the models and record the results

for cls in classifiers:

    model = cls.fit(X_train, y_train)

    ypred = model.predict_proba(X_test)[::,1]

    precision,recall, _ = precision_recall_curve(y_test,  ypred)

    prc_val = prc_values[i]

    result_table1 = result_table1.append({'classifiers':cls.__class__.__name__,

                                        'precision':precision,

                                        'recall':recall,

                                        'prc':prc_val}, ignore_index=True)

    i=i+1

# Set name of the classifiers as index labels

result_table1.set_index('classifiers', inplace=True)

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

for i in result_table.index:

    plt.plot(result_table1.loc[i]['precision'],

             result_table1.loc[i]['recall'

label="{}, PRC={:.3f}".format(i, result_table1.loc[i]['prc']))
```

```
plt.plot([0,1], [0,1], color='orange', linestyle='--')
plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("Recall", fontsize=15)
plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("Precison", fontsize=15)
plt.title('PRC Curve Analysis', fontweight='bold', fontsize=15)
plt.legend(loc='best',fontsize=8)
plt.show()
```

Feature Ranking:

```
features=['age', 'sex', 'cp', 'trestbps', 'chol', 'fbs', 'restecg', 'thalach', 'exang', 'oldpeak', 'slope', 'ca', 'thal']
```

AdaBoostM1:

```
from prettytable import PrettyTable
feature_importance={}
ab_classifier.fit(X, y)
importance = ab_classifier.feature_importances_
metric_table = PrettyTable(["Feature Name", 'Co-effecient Score'])
for i,v in enumerate(importance):
    metric_table.add_row([features[i], round(v, 5)])
    feature_importance[features[i]]=v
print(metric_table)
feature_importance=sorted(feature_importance.items(), key=lambda x:x[1],reverse=True)
fig, ax = plt.subplots(figsize=(3,3))
sns.barplot(y=[feature_importance[i][0] for i in range(len(feature_importance)) ], x=[feature_importance[i][1] for i in range(len(feature_importance)) ],palette='Spectral')
```

Logistic Regression:

```
from prettytable import PrettyTable
feature_importance={}
```

```
lr_classifier.fit(X, y)

importance = lr_classifier.coef_[0]

metric_table = PrettyTable(["Feature Name", 'Co-effecient Score'])

for i,v in enumerate(importance):

    metric_table.add_row([features[i], round(v, 5)])

    feature_importance[features[i]]=v

print(metric_table)

feature_importance=sorted(feature_importance.items(), key=lambda x:x[1],reverse=True)

fig, ax = plt.subplots(figsize=(3,3))

sns.barplot(y=[feature_importance[i][0] for i in range(len(feature_importance))
],x=[feature_importance[i][1] for i in range(len(feature_importance)) ],palette='Spectral')
```

Decision Tree:

```
from prettytable import PrettyTable

feature_importance={}

dt_classifier.fit(X, y)

importance = dt_classifier.feature_importances_

metric_table = PrettyTable(["Feature Name", 'Co-effecient Score'])

for i,v in enumerate(importance):

    metric_table.add_row([features[i], round(v, 5)])

    feature_importance[features[i]]=v

print(metric_table)

feature_importance=sorted(feature_importance.items(), key=lambda x:x[1],reverse=True)

fig, ax = plt.subplots(figsize=(3,3))

sns.barplot(y=[feature_importance[i][0] for i in range(len(feature_importance))
],x=[feature_importance[i][1] for i in range(len(feature_importance)) ],palette='Spectral')
```

Random Forest:

```
from prettytable import PrettyTable

feature_importance={}

rf_classifier.fit(X, y)
```

```
importance = rf_classifier.feature_importances_
metric_table = PrettyTable(["Feature Name", 'Co-effecient Score'])
for i,v in enumerate(importance):
    metric_table.add_row([features[i], round(v, 5)])
    feature_importance[features[i]]=v
print(metric_table)
feature_importance=sorted(feature_importance.items(), key=lambda x:x[1],reverse=True)
fig, ax = plt.subplots(figsize=(3,3))
sns.barplot(y=[feature_importance[i][0] for i in range(len(feature_importance)) ],
x=[feature_importance[i][1] for i in range(len(feature_importance)) ],palette='Spectral'
```

CHAPTER 4

SNAPSHOTS

```
age         False
sex         False
cp          False
trestbps    False
chol        False
fbs         False
restecg     False
thalach     False
exang       False
oldpeak     False
slope       False
ca          False
thal        False
target      False
dtype: bool
```

Figure 4.1 – Checking Null Values in the Dataset

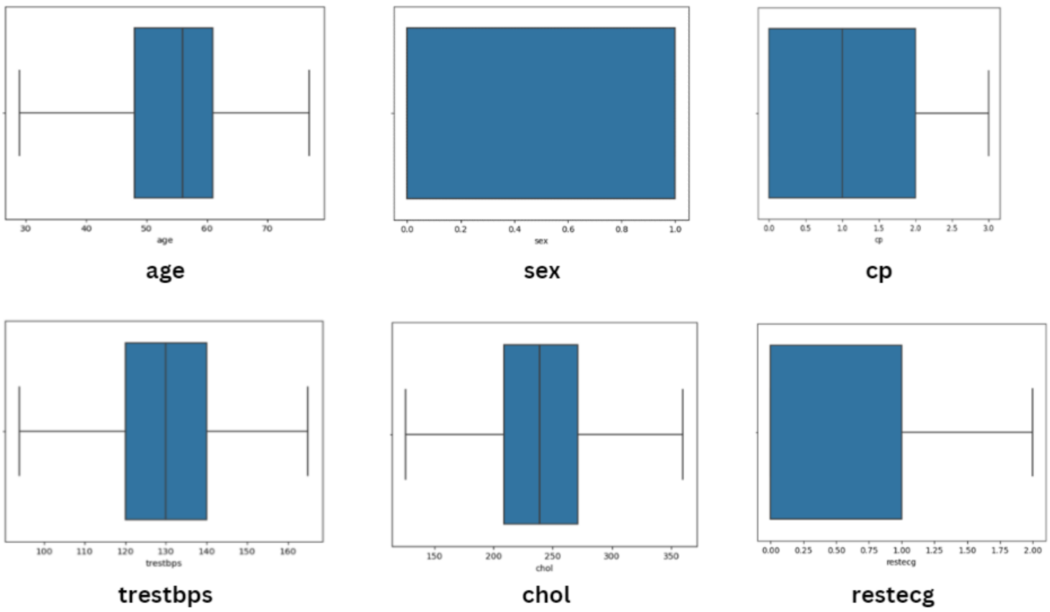


Figure 4.2 – Attribute Boxplot after Outlier Removal

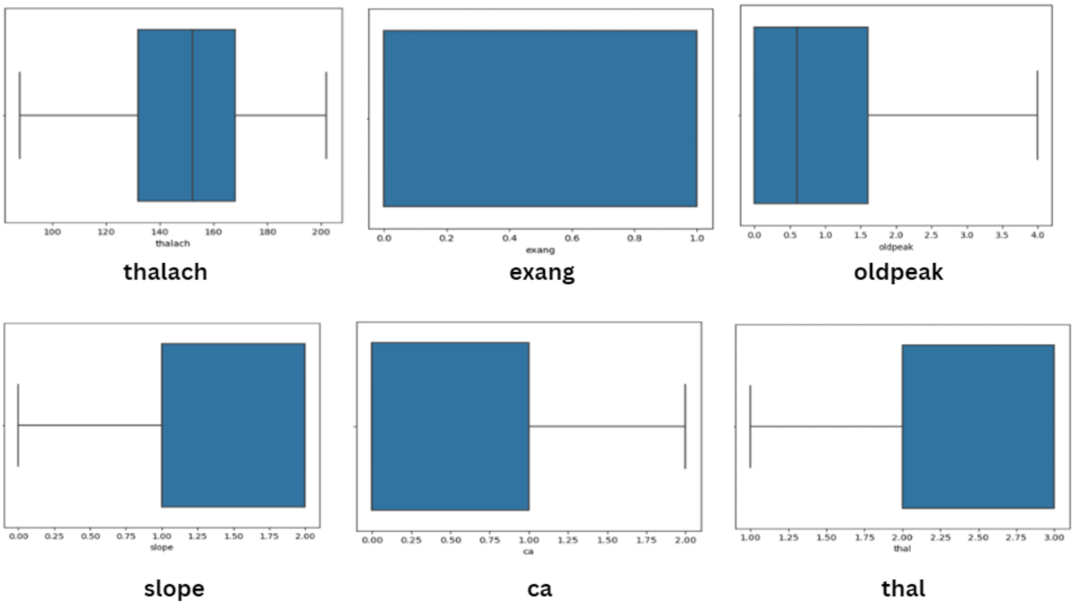


Figure 4.3 – Attribute Boxplot after Outlier Removal

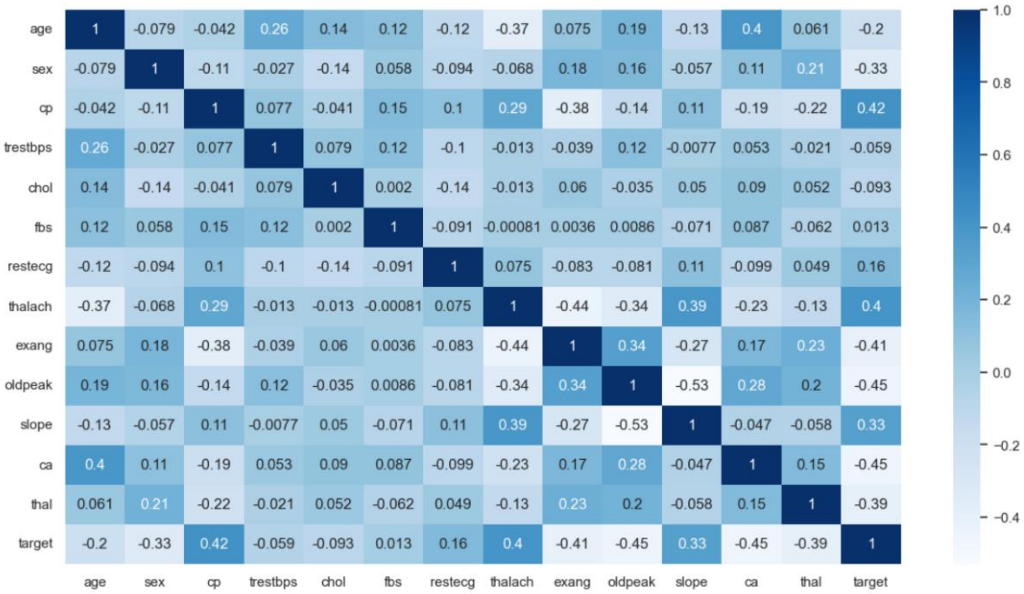


Figure 4.4 – Heatmap showing correlation among attributes

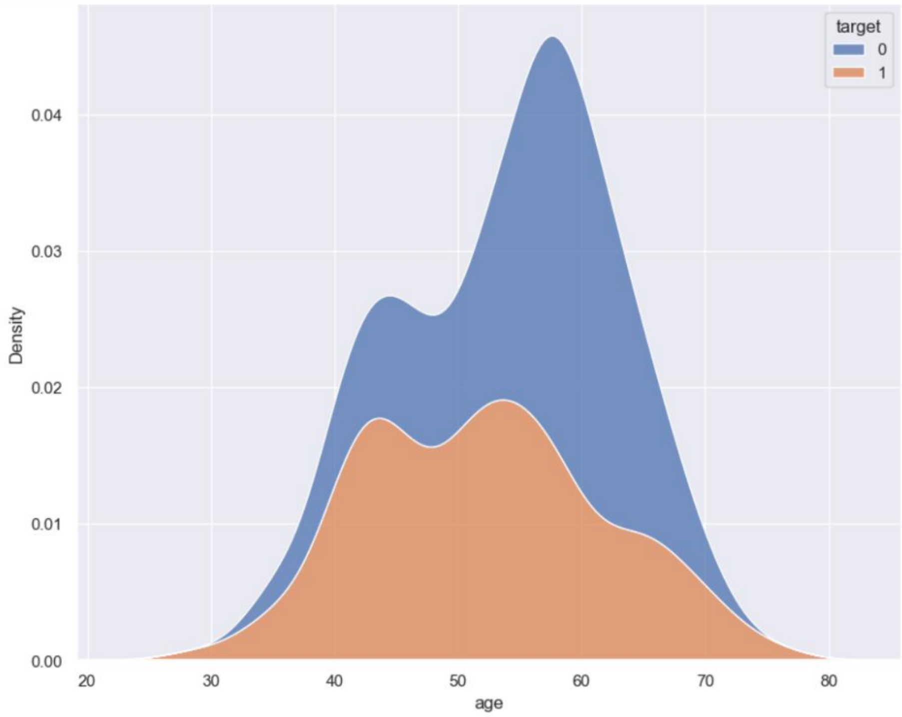


Figure 4.5 – KDE plot

Model Name	Accuracy	Recall	Precision	F1-score	MCC-Score	kappa-score	sensitivity	specificity	AUROC	AUPRC
AdaBoostM1	99.792	1.0	0.996	0.998	0.996	0.996	1.0	0.996	0.998	0.996

Figure 4.6 – AdaBoostM1 Result Metrics

Model Name	Accuracy	Recall	Precision	F1-score	MCC-Score	kappa-score	sensitivity	specificity	AUROC	AUPRC
KNN	100.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0

Figure 4.7 – K-Nearest Neighbors Result Metrics

Model Name	Accuracy	Recall	Precision	F1-score	MCC-Score	kappa-score	sensitivity	specificity	AUROC	AUPRC
LR	86.271	0.902	0.839	0.868	0.726	0.72	0.902	0.82	0.861	0.807

Figure 4.8 – Logistic Regression Result Metrics

Model Name	Accuracy	Recall	Precision	F1-score	MCC-Score	kappa-score	sensitivity	specificity	AUROC	AUPRC
DT	99.372	1.0	0.988	0.994	0.988	0.987	1.0	0.987	0.993	0.988

Figure 4.9– Decision Tree Result Metrics

Model Name	Accuracy	Recall	Precision	F1-score	MCC-Score	kappa-score	sensitivity	specificity	AUROC	AUPRC
RF	99.581	1.0	0.992	0.996	0.992	0.991	1.0	0.992	0.996	0.992

Figure 4.10 – Random Forest Result Metrics

Model Name	Accuracy	Recall	Precision	F1-score	MCC-Score	kappa-score	sensitivity	specificity	AUROC	AUPRC
MLP	95.175	0.947	0.958	0.952	0.902	0.899	0.947	0.945	0.946	0.933

Figure 4.11 – Multilayer Perceptron Result Metrics

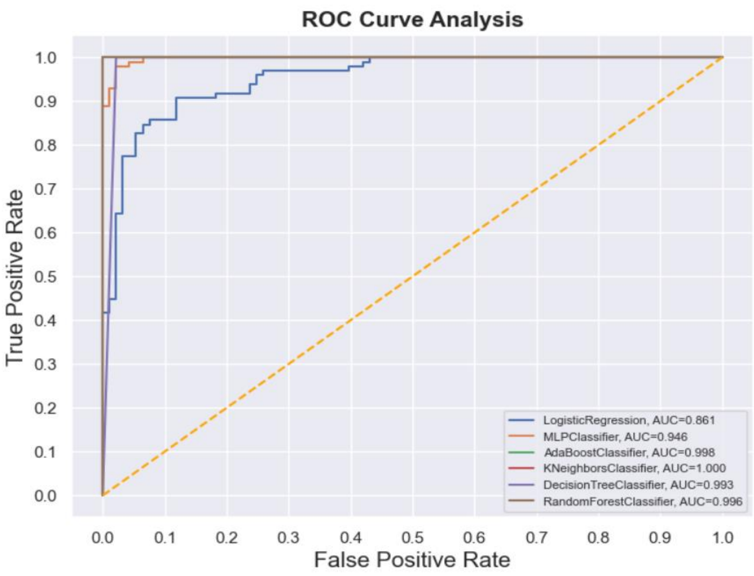


Figure 4.12 – AUROC Plot

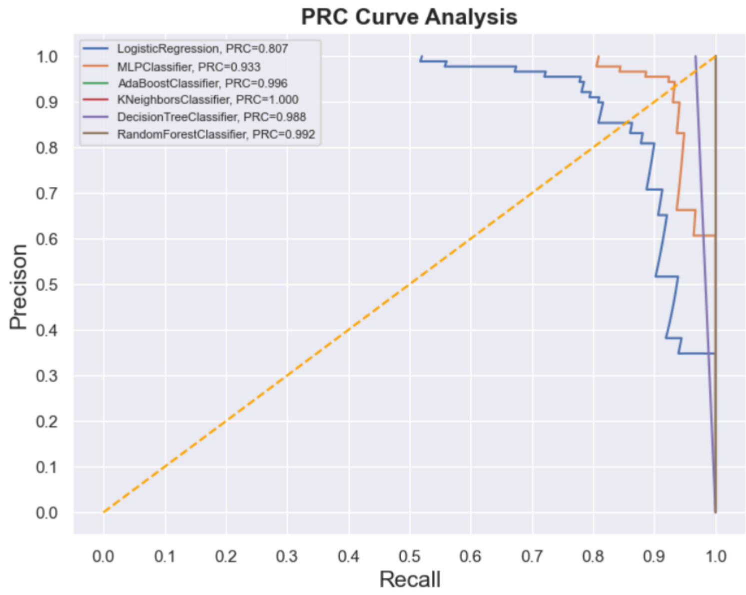
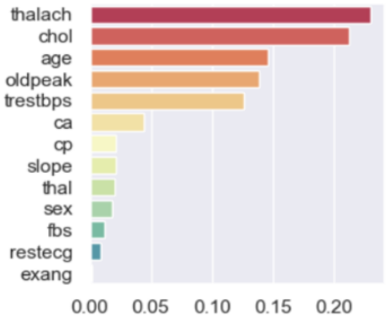
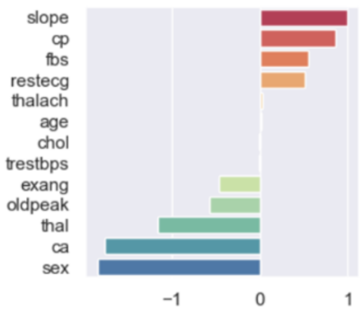


Figure 4.13 – AUPRC Plot



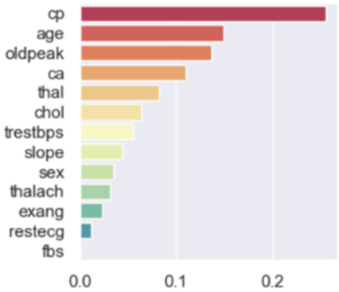
Feature Name	Co-effecient Score
age	0.14571
sex	0.01857
cp	0.02143
trestbps	0.12571
chol	0.21286
fbs	0.01143
restecg	0.00857
thalach	0.23
exang	0.00143
oldpeak	0.13857
slope	0.02143
ca	0.04429
thal	0.02

AdaBoostM1



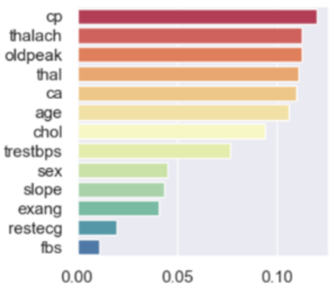
Feature Name	Co-effecient Score
age	0.01881
sex	-1.84579
cp	0.85817
trestbps	-0.01726
chol	-0.00825
fbs	0.55047
restecg	0.50645
thalach	0.02947
exang	-0.46796
oldpeak	-0.56756
slope	0.98755
ca	-1.7624
thal	-1.16801

Logistic Regression



Feature Name	Co-effecient Score
age	0.14892
sex	0.03487
cp	0.25538
trestbps	0.05633
chol	0.0643
fbs	0.0
restecg	0.01238
thalach	0.03106
exang	0.02369
oldpeak	0.13621
slope	0.04461
ca	0.11019
thal	0.08206

Decision Tree



Feature Name	Co-effecient Score
age	0.10559
sex	0.04522
cp	0.11978
trestbps	0.07645
chol	0.09396
fbs	0.01083
restecg	0.01956
thalach	0.11219
exang	0.04064
oldpeak	0.11199
slope	0.04367
ca	0.10933
thal	0.11079

Random Forest

Figure 4.14– Feature Ranking

CHAPTER 5

CONCLUSION AND FUTURE PLANS

Machine Learning's potential for accurate disease prediction has pushed us to implement it in the field of cardiovascular prediction. Here, we used a heart disease dataset downloaded from Kaggle and applied multiple supervised Machine Learning algorithms to predict the presence or absence of heart disease. KNN performed extremely well giving us 100% accuracy. ABM1, DT and RF also produced good results with accuracies in the range 99-100%. While MLP gave an accuracy of 95.175%, LR gave the least accuracy with a value of 86.271%. The accuracies were calculated by applying 10-fold cross-validation. In addition, AUROC and AUPRC curves were plotted to understand the overall performance of the binary classifier and summary of binary classifier's overall predictive performance. Finally, feature ranking was done for all algorithms except Multilayer Perceptron (MLP) and K-Nearest Neighbor (KNN). These features were ranked according to their feature importance scores.

In future, the same work can be extended to be implemented for relatively large datasets with more samples and more features. Also, other Supervised machine learning techniques like SVM can be implemented using different Kernel functions. An ensemble approach by combining the classifiers can be tested to make the predictions more precise and accurate.

CHAPTER 6

REFERENCES

[1] Heart Disease Dataset used for implementing this project : https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset

[2] Md Mamun Ali, Bikash Kumar Paul, Kawsar Ahmed, Francis M. Bui, Julian M.W. Quinn, Mohammad Ali Moni, Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison, *Computers in Biology and Medicine, Volume 136, 2021, 104672, ISSN 0010-4825,* https://doi.org/10.1016/j.compbiomed.2021.104672 .

[3] Soni, Jyoti & Ansari, Ujma & Sharma, Dipesh & Soni, Sunita. (2011). Predictive Data Mining for Medical Diagnosis: An Overview of Heart Disease Prediction. International *Journal of Computer Applications. 17. 43-48. 10.5120/2237-2860.*

[4] Jan M, Awan AA, Khalid MS, Nisar S. Ensemble approach for developing a smart heart disease prediction system using classification algorithms. *Research Reports in Clinical Cardiology*. *2018;9:33-45* https://doi.org/10.2147/RRCC.S172035 .

[5] H. M. LE, T. D. TRAN, and L. V. TRAN, "AUTOMATIC HEART DISEASE PREDICTION USING FEATURE SELECTION AND DATA MINING TECHNIQUE", *JCC, vol. 34, no. 1, p. 33–48, Aug. 2018*.

[6] S. Mohan, C. Thirumalai and G. Srivastava, "Effective Heart Disease Prediction Using Hybrid Machine Learning Techniques," *in IEEE Access, vol. 7, pp. 81542-81554, 2019,* doi: 10.1109/ACCESS.2019.2923707.

[7] C. Beulah Christalin Latha, S. Carolin Jeeva, Improving the accuracy of prediction of heart disease risk based on ensemble classification techniques, *Informatics in Medicine Unlocked, Volume 16, 2019, 100203, ISSN 2352-9148,* https://doi.org/10.1016/j.imu.2019.100203.

CHAPTER 7

APPENDIX OF BASE PAPER

Contents lists available at ScienceDirect

Computers in Biology and Medicine

journal homepage: www.elsevier.com/locate/compbiomed

Heart disease prediction using supervised machine learning algorithms: Performance analysis and comparison

Md Mamun Ali [a], Bikash Kumar Paul [a,b,c], Kawsar Ahmed [b,c,*], Francis M. Bui [d], Julian M. W. Quinn [e], Mohammad Ali Moni [e,f,**]

[a] Department of Software Engineering (SWE), Daffodil International University (DIU), Sukrabad, Dhaka, 1207, Bangladesh
[b] Group of Biophotomatiχ, Department of Information and Communication Technology, Mawlana Bhashani Science and Technology University, Santosh, Tangail-1902, Bangladesh
[c] Department of Information and Communication Technology, Mawlana Bhashani Science and Technology University, Santosh, Tangail, 1902, Bangladesh
[d] Department of Electrical and Computer Engineering, University of Saskatchewan, 57 Campus Drive, Saskatoon, SK S7N 5A9, Canada
[e] Healthy Ageing Theme, Garvan Institute of Medical Research, Darlinghurst, NSW, 2010, Australia
[f] WHO Collaborating Centre on EHealth, UNSW Digital Health, School of Public Health and Community Medicine, Faculty of Medicine, UNSW Sydney, NSW 2052, Australia

ARTICLE INFO

Keywords:
Cardiovascular disease
Machine learning
Random forest
Decision tree
KNN

ABSTRACT

Machine learning and data mining-based approaches to prediction and detection of heart disease would be of great clinical utility, but are highly challenging to develop. In most countries there is a lack of cardiovascular expertise and a significant rate of incorrectly diagnosed cases which could be addressed by developing accurate and efficient early-stage heart disease prediction by analytical support of clinical decision-making with digital patient records. This study aimed to identify machine learning classifiers with the highest accuracy for such diagnostic purposes. Several supervised machine-learning algorithms were applied and compared for performance and accuracy in heart disease prediction. Feature importance scores for each feature were estimated for all applied algorithms except MLP and KNN. All the features were ranked based on the importance score to find those giving high heart disease predictions. This study found that using a heart disease dataset collected from Kaggle three-classification based on k-nearest neighbor (KNN), decision tree (DT) and random forests (RF) algorithms the RF method achieved 100% accuracy along with 100% sensitivity and specificity. Thus, we found that a relatively simple supervised machine learning algorithm can be used to make heart disease predictions with very high accuracy and excellent potential utility.

Figure 7.1– Base Paper Appendix