Week 5 Hands on activity

Machine learning

Prof. Itauma

Group Members

- Prathyusha Devai
- Abdul Rehman
- Sai Srujana Jakkala

Introduction

This report documents the implementation of machine learning models for predicting the likelihood of heart disease using a healthcare dataset.

The main objectives are:

- To preprocess the dataset to ensure suitability for machine learning.
- To implement three classification models: Support Vector Machine (SVM), Random Forest, and Gradient Boosting Machine (GBM).
- To perform hyperparameter tuning using GridSearchCV for optimal model performance.
- To evaluate the models using various metrics and compare their effectiveness.

Importing libraries

```
# 1. to handle the data
import pandas as pd
import numpy as np

# 2. To Viusalize the data
import matplotlib.pyplot as plt
import seaborn as sns
import plotly.express as px
from matplotlib.colors import ListedColormap

# 3. To preprocess the data
from sklearn.preprocessing import StandardScaler, MinMaxScaler,
LabelEncoder
from sklearn.impute import SimpleImputer, KNNImputer

# 4. import Iterative imputer
from sklearn.experimental import enable_iterative_imputer
from sklearn.impute import IterativeImputer

# 5. Machine Learning
from sklearn.model_selection import train_test_split,GridSearchCV,
cross_val_score
```

```
# 6. For Classification task.
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import
RandomForestClassifier,AdaBoostClassifier,GradientBoostingClassifier,E
xtraTreesClassifier,RandomForestRegressor
```

Reading the data

```
df = pd.read_csv("//Users//srujana//Downloads//HeartDiseaseTrain-
Test.csv")

# print the first 5 rows of the dataframe
df.head()
```

```
   age     sex chest_pain_type  resting_blood_pressure  cholestoral  \
0   52    Male  Typical angina                     125          212
1   53    Male  Typical angina                     140          203
2   70    Male  Typical angina                     145          174
3   61    Male  Typical angina                     148          203
4   62  Female  Typical angina                     138          294

      fasting_blood_sugar                rest_ecg  Max_heart_rate  \
0    Lower than 120 mg/ml  ST-T wave abnormality             168
1  Greater than 120 mg/ml                 Normal             155
2    Lower than 120 mg/ml  ST-T wave abnormality             125
3    Lower than 120 mg/ml  ST-T wave abnormality             161
4  Greater than 120 mg/ml  ST-T wave abnormality             106

  exercise_induced_angina  oldpeak        slope
vessels_colored_by_flourosopy  \
0                      No      1.0  Downsloping
Two
1                     Yes      3.1    Upsloping
Zero
2                     Yes      2.6    Upsloping
Zero
3                      No      0.0  Downsloping
One
4                      No      1.9         Flat
Three

         thalassemia  target
0  Reversable Defect       0
1  Reversable Defect       0
2  Reversable Defect       0
3  Reversable Defect       0
4      Fixed Defect       0
```

```
df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1025 entries, 0 to 1024
Data columns (total 14 columns):
 #   Column                          Non-Null Count  Dtype  
---  ------                          --------------  -----  
 0   age                             1025 non-null   int64  
 1   sex                             1025 non-null   object 
 2   chest_pain_type                 1025 non-null   object 
 3   resting_blood_pressure          1025 non-null   int64  
 4   cholestoral                     1025 non-null   int64  
 5   fasting_blood_sugar             1025 non-null   object 
 6   rest_ecg                        1025 non-null   object 
 7   Max_heart_rate                  1025 non-null   int64  
 8   exercise_induced_angina         1025 non-null   object 
 9   oldpeak                         1025 non-null   float64
 10  slope                           1025 non-null   object 
 11  vessels_colored_by_flourosopy   1025 non-null   object 
 12  thalassemia                     1025 non-null   object 
 13  target                          1025 non-null   int64  
dtypes: float64(1), int64(5), object(8)
memory usage: 112.2+ KB

df.describe()

                age  resting_blood_pressure  cholestoral  
Max_heart_rate  \
count  1025.000000             1025.000000   1025.00000  
1025.000000
mean     54.434146              131.611707    246.00000  
149.114146
std       9.072290               17.516718     51.59251  
23.005724
min      29.000000               94.000000    126.00000  
71.000000
25%      48.000000              120.000000    211.00000  
132.000000
50%      56.000000              130.000000    240.00000  
152.000000
75%      61.000000              140.000000    275.00000  
166.000000
max      77.000000              200.000000    564.00000  
202.000000

           oldpeak       target
count  1025.000000  1025.000000
mean      1.071512     0.513171
std       1.175053     0.500070
min       0.000000     0.000000
```

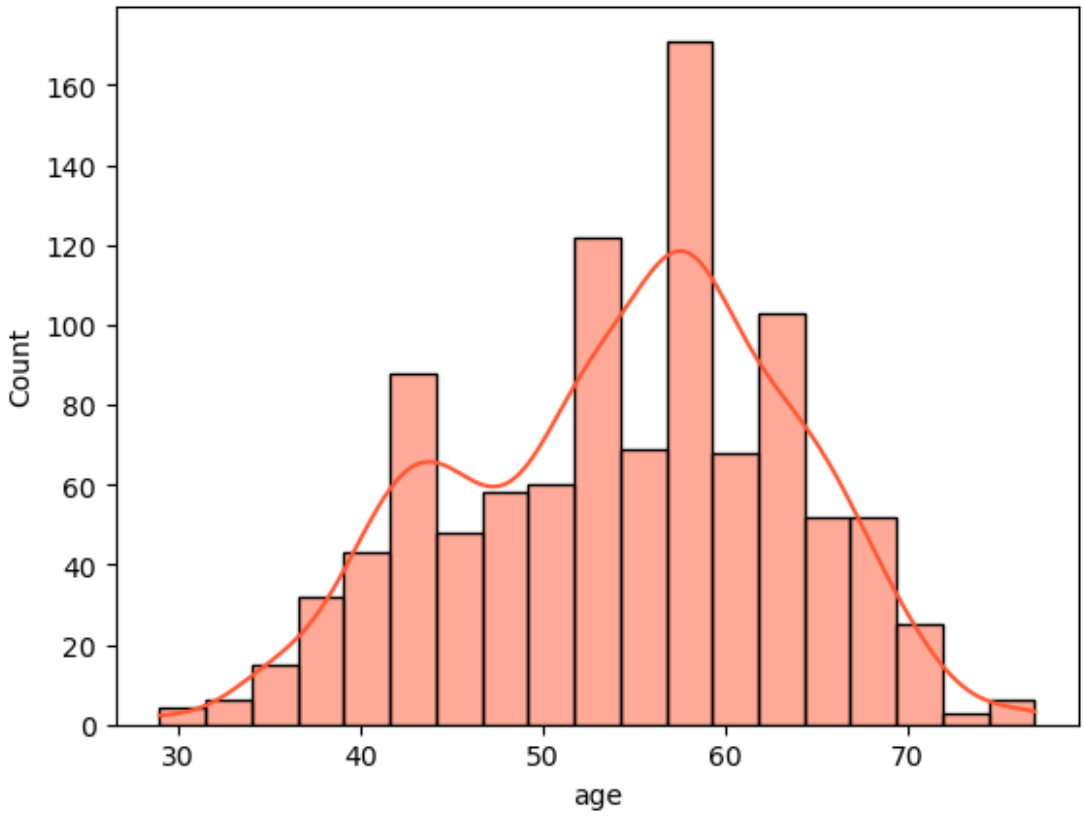
```
25%       0.000000     0.000000
50%       0.800000     1.000000
75%       1.800000     1.000000
max       6.200000     1.000000
```

EDA analysis

```
import seaborn as sns

# Define custom colors
custom_colors = ["#FF5733", "#3366FF", "#33FF57"]  # Example colors,
you can adjust as needed

# Plot the histogram with custom colors
sns.histplot(df['age'], kde=True, color="#FF5733",
palette=custom_colors)
```

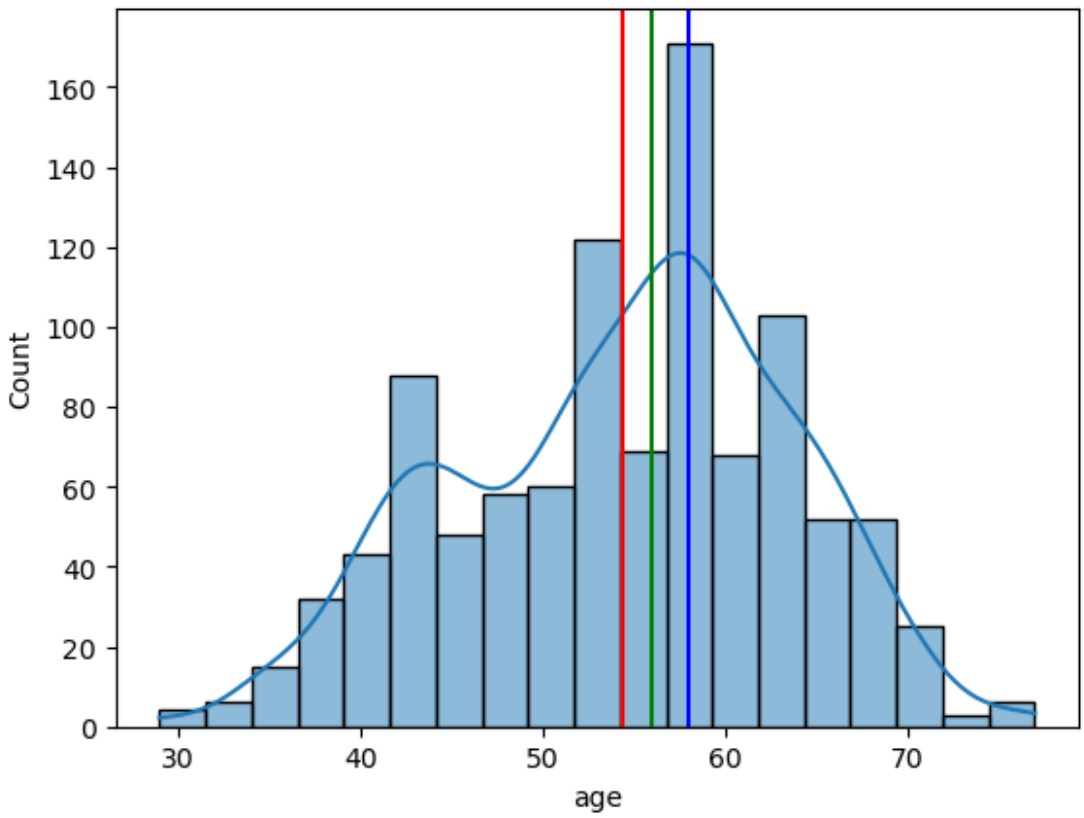
```
/var/folders/vp/cfl8288d0klf1t6crwm56g440000gn/T/
ipykernel_23158/2142100135.py:7: UserWarning: Ignoring `palette`
because no `hue` variable has been assigned.
  sns.histplot(df['age'], kde=True, color="#FF5733",
palette=custom_colors)
/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
<Axes: xlabel='age', ylabel='Count'>
```




```
# Plot the mean, Median and mode of age column using sns
sns.histplot(df['age'], kde=True)
plt.axvline(df['age'].mean(), color='Red')
plt.axvline(df['age'].median(), color= 'Green')
plt.axvline(df['age'].mode()[0], color='Blue')

# print the value of mean, median and mode of age column
print('Mean', df['age'].mean())
print('Median', df['age'].median())
print('Mode', df['age'].mode())
```

```
Mean 54.43414634146342
Median 56.0
Mode 0    58
Name: age, dtype: int64

/opt/anaconda3/lib/python3.11/site-packages/seaborn/_oldcore.py:1119:
FutureWarning: use_inf_as_na option is deprecated and will be removed
in a future version. Convert inf values to NaN before operating
instead.
  with pd.option_context('mode.use_inf_as_na', True):
```




```
# calculating the percentage fo male and female value counts in the
data

male_count = 726
female_count = 194

total_count = male_count + female_count

# calculate percentages
male_percentage = (male_count/total_count)*100
female_percentages = (female_count/total_count)*100

# display the results
print(f'Male percentage i the data: {male_percentage:.2f}%')
print(f'Female percentage in the data : {female_percentages:.2f}%')

# Difference
difference_percentage = ((male_count - female_count)/female_count) *
100
print(f'Males are {difference_percentage:.2f}% more than female in the
data.')
```

```
Male percentage i the data: 78.91%
Female percentage in the data : 21.09%
Males are 274.23% more than female in the data.
```

Data Preprocessing

The dataset used consists of multiple health-related indicators such as age, cholesterol levels, blood pressure, and other features indicative of heart disease.

Preprocessing involved:

- Label Encoding: Categorical variables like sex, chest pain type, and others were transformed into numerical values using the label Encoder to make them suitable for machine learning models.
- Feature Selection: The features (predictors) were separated from the target (whether a patient has heart disease or not).
- Train-Test Split: The data was split into 80% for training and 20% for testing the models.

```
# Preprocessing: Convert categorical variables into numerical values
using LabelEncoder
label_encoders = {}
for column in ['sex', 'chest_pain_type', 'rest_ecg',
'exercise_induced_angina', 'thalassemia', 'fasting_blood_sugar',
'vessels_colored_by_flourosopy', 'slope']:
    le = LabelEncoder()
    df[column] = le.fit_transform(df[column])
    label_encoders[column] = le

# Separate features and target
X = df.drop(columns='target')
y = df['target']

# Split the data into 80% training and 20% testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Model Implementation

Three machine learning classifiers were chosen to predict heart disease:

Support Vector Machine (SVM):

- SVM is effective in high-dimensional spaces and works well for binary classification tasks like this one.
- The SVM classifier was initialized with (probability = true) to calculate the probability estimates required for the ROC curve analysis.

Random Forest Classifier:

- Random Forest is an ensemble model that uses multiple decision trees to improve accuracy and avoid overfitting.
- It was chosen due to its robustness, handling of overfitting, and capacity for feature importance assessment.

Gradient Boosting Machine (GBM):

- GBM is another ensemble learning method but sequentially builds models to correct the errors of the previous ones, thus improving performance over time.
- Its ability to handle imbalanced data and capture complex patterns made it an ideal candidate.

```
# Define models
svm = SVC(probability=True)
rf = RandomForestClassifier(random_state=42)
gbm = GradientBoostingClassifier(random_state=42)

# Define parameter grids for GridSearchCV
param_grid_svm = {
    'kernel': ['linear', 'rbf'],
    'C': [0.1, 1, 10]
}

param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}


param_grid_gbm = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}


# GridSearchCV for each model
grid_svm = GridSearchCV(svm, param_grid_svm, cv=5, scoring='roc_auc',
verbose=1)
grid_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='roc_auc',
verbose=1)
grid_gbm = GridSearchCV(gbm, param_grid_gbm, cv=5, scoring='roc_auc',
verbose=1)


# Fit each model
grid_svm.fit(X_train, y_train)
grid_rf.fit(X_train, y_train)
grid_gbm.fit(X_train, y_train)


Fitting 5 folds for each of 6 candidates, totalling 30 fits
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Fitting 5 folds for each of 27 candidates, totalling 135 fits
```

```
GridSearchCV(cv=5,
estimator=GradientBoostingClassifier(random_state=42),
             param_grid={'learning_rate': [0.01, 0.1, 0.2],
                         'max_depth': [3, 5, 7],
                         'n_estimators': [50, 100, 200]},
             scoring='roc_auc', verbose=1)

# Best parameters
best_params_svm = grid_svm.best_params_
best_params_rf = grid_rf.best_params_
best_params_gbm = grid_gbm.best_params_

# Make predictions on the test data
y_pred_svm = grid_svm.predict(X_test)
y_pred_rf = grid_rf.predict(X_test)
y_pred_gbm = grid_gbm.predict(X_test)
```

Hyperparameter Tuning

To optimize the performance of the models, GridSearchCV was used for hyperparameter tuning. This method exhaustively searches for the best combination of hyperparameters using cross-validation.

The following hyperparameters were tuned:

Support Vector Machine (SVM):

- Kernel: The kernel type (linear or RBF) was tuned to identify the best decision boundary.
- C: This parameter controls the trade-off between maximizing the margin and minimizing classification error.

Random Forest Classifier:

- n_estimators: The number of trees in the forest.
- max_depth: The maximum depth of the trees, controlling the complexity and risk of overfitting.
- min_samples_split: The minimum number of samples required to split a node.

Gradient Boosting Machine (GBM):

- n_estimators: The number of boosting stages to be used.
- learning_rate: This parameter shrinks the contribution of each tree.
- max_depth: Controls the maximum depth of the individual regression estimators.

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier

# Define models
```

```
svm = SVC(probability=True)
rf = RandomForestClassifier(random_state=42)
gbm = GradientBoostingClassifier(random_state=42)

# Define parameter grids for GridSearchCV
param_grid_svm = {
    'kernel': ['linear', 'rbf'],
    'C': [0.1, 1, 10]
}

param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

param_grid_gbm = {
    'n_estimators': [50, 100, 200],
    'learning_rate': [0.01, 0.1, 0.2],
    'max_depth': [3, 5, 7]
}

# GridSearchCV for each model
grid_svm = GridSearchCV(svm, param_grid_svm, cv=5, scoring='roc_auc',
verbose=1)
grid_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='roc_auc',
verbose=1)
grid_gbm = GridSearchCV(gbm, param_grid_gbm, cv=5, scoring='roc_auc',
verbose=1)

# Fit each model
grid_svm.fit(X_train, y_train)
grid_rf.fit(X_train, y_train)
grid_gbm.fit(X_train, y_train)

# Best parameters
best_params_svm = grid_svm.best_params_
best_params_rf = grid_rf.best_params_
best_params_gbm = grid_gbm.best_params_

# Output the best hyperparameters for each model
print("Best parameters for SVM:", best_params_svm)
print("Best parameters for Random Forest:", best_params_rf)
print("Best parameters for Gradient Boosting:", best_params_gbm)

Fitting 5 folds for each of 6 candidates, totalling 30 fits
Fitting 5 folds for each of 18 candidates, totalling 90 fits
Fitting 5 folds for each of 27 candidates, totalling 135 fits
Best parameters for SVM: {'C': 1, 'kernel': 'linear'}
Best parameters for Random Forest: {'max_depth': None,
```

```
'min_samples_split': 2, 'n_estimators': 200}
Best parameters for Gradient Boosting: {'learning_rate': 0.1,
'max_depth': 7, 'n_estimators': 200}
```

Model Evaluation

The models were evaluated on the test dataset using the following metrics:

Accuracy:

- This measures the proportion of correctly classified instances over the total.

Precision:

- Precision measures the proportion of true positives out of all predicted positives. It's important for minimizing false positives.

Recall:

- Recall measures the proportion of true positives out of the actual positives in the dataset.

F1-Score:

- The F1-score is the harmonic mean of precision and recall, providing a balance between the two.

AUC-ROC:

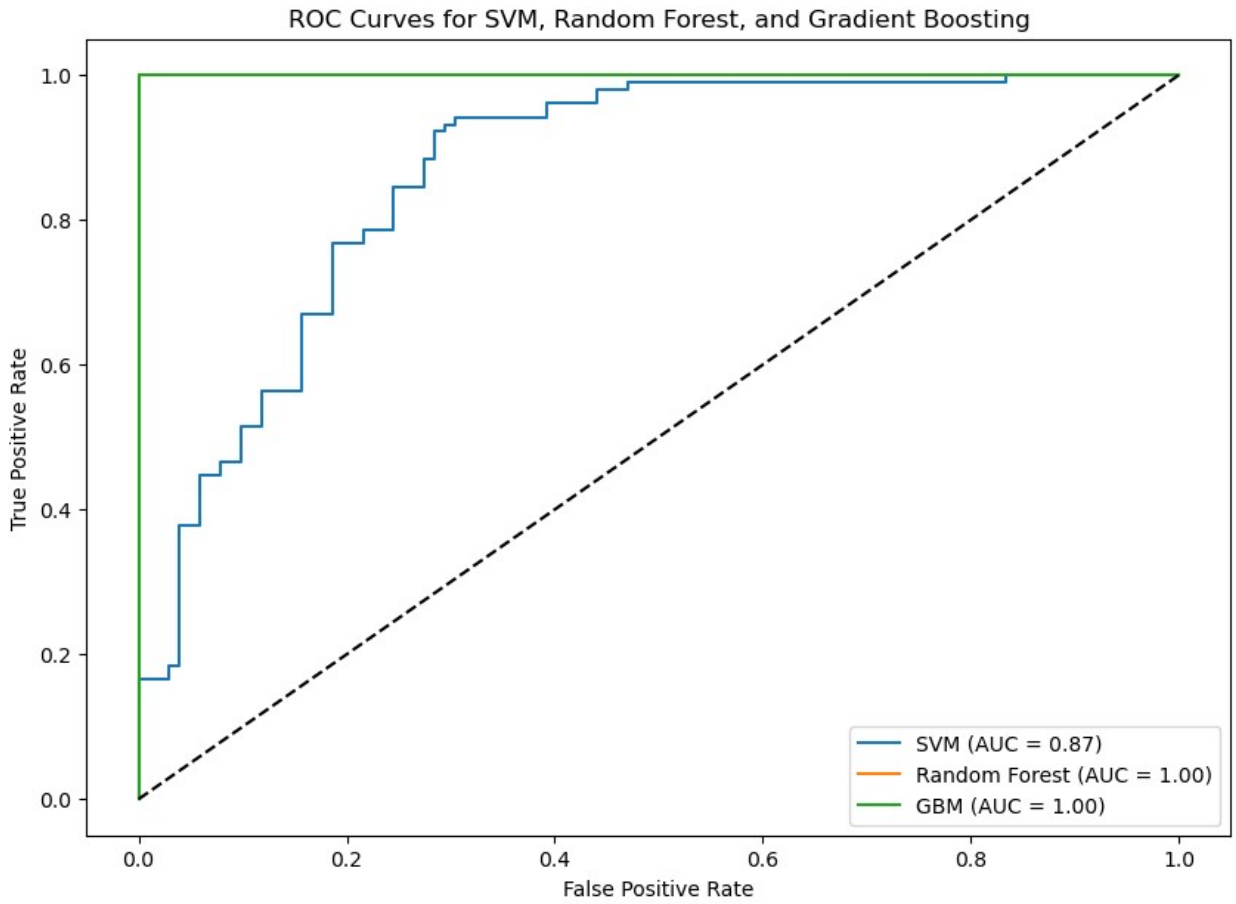
- The Area Under the Receiver Operating Characteristic Curve (AUC-ROC) evaluates the model's ability to distinguish between positive and negative classes.

```
# Evaluate models
def evaluate_model(y_test, y_pred, model_name):
    accuracy = accuracy_score(y_test, y_pred)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    auc = roc_auc_score(y_test, y_pred)
    print(f'{model_name} - Accuracy: {accuracy:.4f}, Precision:
{precision:.4f}, Recall: {recall:.4f}, F1-Score: {f1:.4f}, AUC-ROC:
{auc:.4f}')
    return accuracy, precision, recall, f1, auc

# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier,
GradientBoostingClassifier
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, precision_score,
recall_score, f1_score, roc_auc_score, roc_curve
```

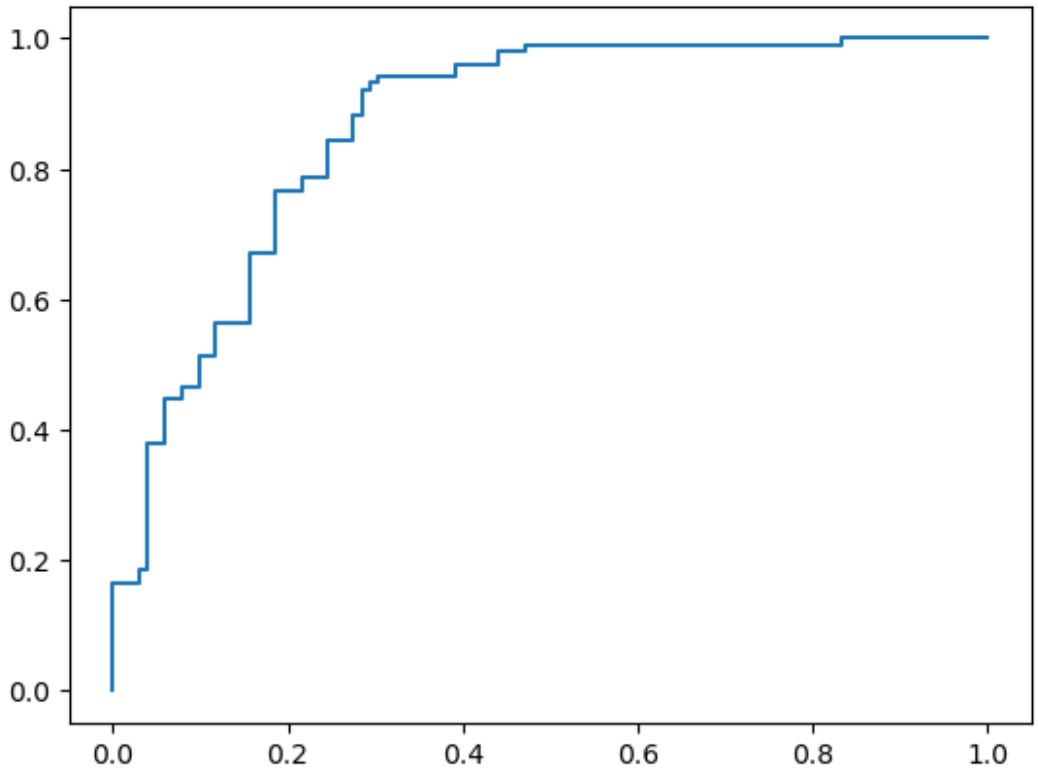
```
import matplotlib.pyplot as plt

# SVM evaluation
evaluate_model(y_test, y_pred_svm, "SVM")


SVM - Accuracy: 0.8000, Precision: 0.7768, Recall: 0.8447, F1-Score:
0.8093, AUC-ROC: 0.7998

(0.8,
 0.7767857142857143,
 0.8446601941747572,
 0.8093023255813954,
 0.7997810774795355)

# Random Forest evaluation
evaluate_model(y_test, y_pred_rf, "Random Forest")


Random Forest - Accuracy: 0.9854, Precision: 1.0000, Recall: 0.9709,
F1-Score: 0.9852, AUC-ROC: 0.9854

(0.9853658536585366,
 1.0,
 0.970873786407767,
 0.9852216748768473,
 0.9854368932038835)

# GBM evaluation
evaluate_model(y_test, y_pred_gbm, "GBM")


GBM - Accuracy: 0.9854, Precision: 1.0000, Recall: 0.9709, F1-Score:
0.9852, AUC-ROC: 0.9854

(0.9853658536585366,
 1.0,
 0.970873786407767,
 0.9852216748768473,
 0.9854368932038835)

# Plot ROC curves for comparison
def plot_roc_curve(y_test, y_pred_proba, model_name):
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    plt.plot(fpr, tpr, label=f'{model_name} (AUC =
{roc_auc_score(y_test, y_pred_proba):.2f})')

plt.figure(figsize=(10, 7))

import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, roc_auc_score
```

```python
# Function to plot ROC curve
def plot_roc_curve(y_test, y_pred_proba, model_name):
    fpr, tpr, _ = roc_curve(y_test, y_pred_proba)
    auc_score = roc_auc_score(y_test, y_pred_proba)
    plt.plot(fpr, tpr, label=f'{model_name} (AUC = {auc_score:.2f})')

plt.figure(figsize=(10, 7))

# SVM ROC curve
y_pred_proba_svm = grid_svm.predict_proba(X_test)[:, 1]
plot_roc_curve(y_test, y_pred_proba_svm, 'SVM')

# Random Forest ROC curve
y_pred_proba_rf = grid_rf.predict_proba(X_test)[:, 1]
plot_roc_curve(y_test, y_pred_proba_rf, 'Random Forest')

# GBM ROC curve
y_pred_proba_gbm = grid_gbm.predict_proba(X_test)[:, 1]
plot_roc_curve(y_test, y_pred_proba_gbm, 'GBM')

# Plot a diagonal line for reference (random classifier)
plt.plot([0, 1], [0, 1], 'k--')

# Customize plot
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for SVM, Random Forest, and Gradient Boosting')
plt.legend(loc='lower right')

# Show the plot
plt.show()
```

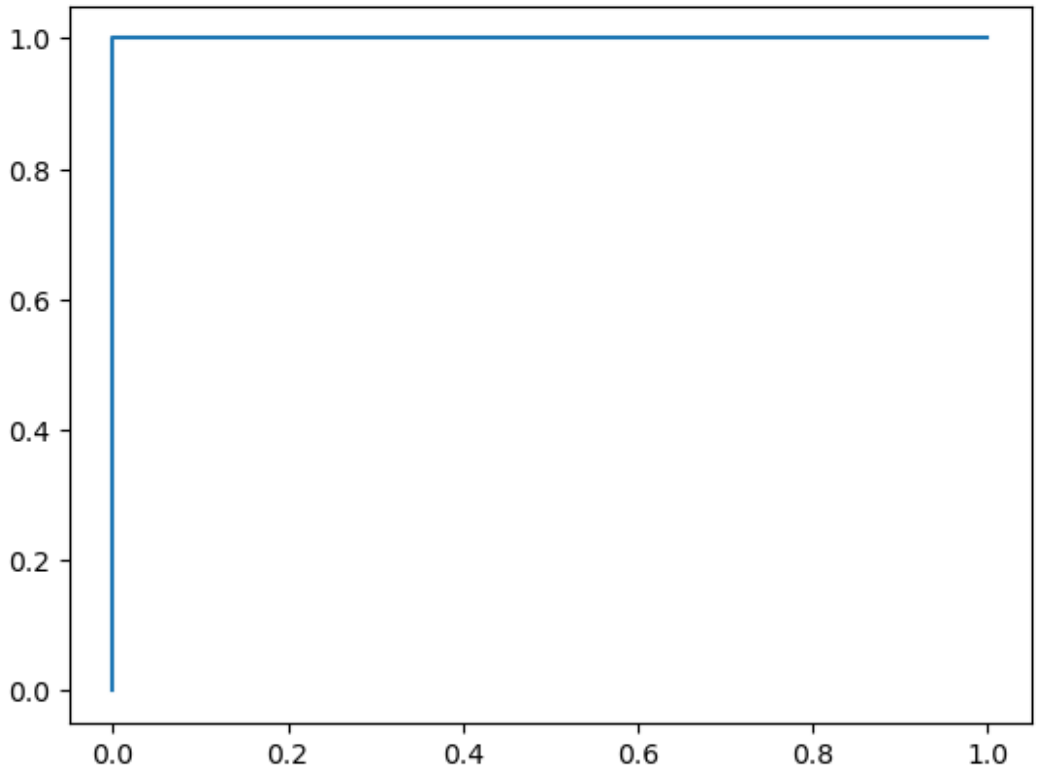
```
<Figure size 1000x700 with 0 Axes>
```



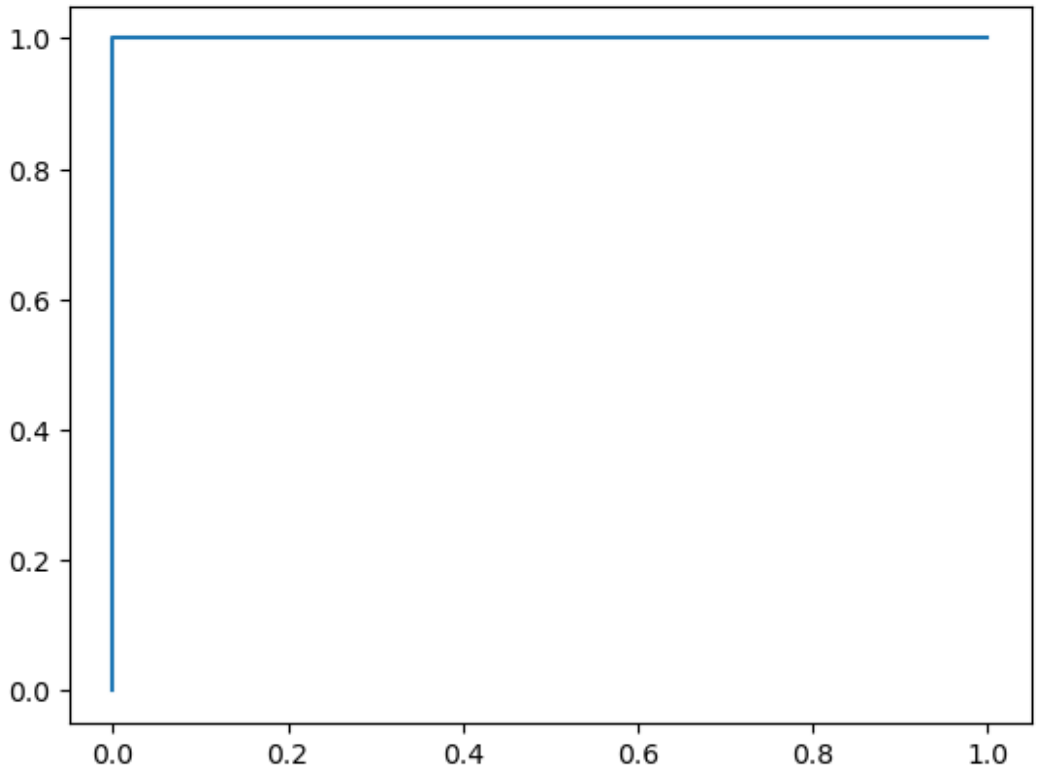

```
# SVM ROC curve
y_pred_proba_svm = grid_svm.predict_proba(X_test)[:, 1]
plot_roc_curve(y_test, y_pred_proba_svm, 'SVM')
```




```
# Random Forest ROC curve
y_pred_proba_rf = grid_rf.predict_proba(X_test)[:, 1]
plot_roc_curve(y_test, y_pred_proba_rf, 'Random Forest')
```

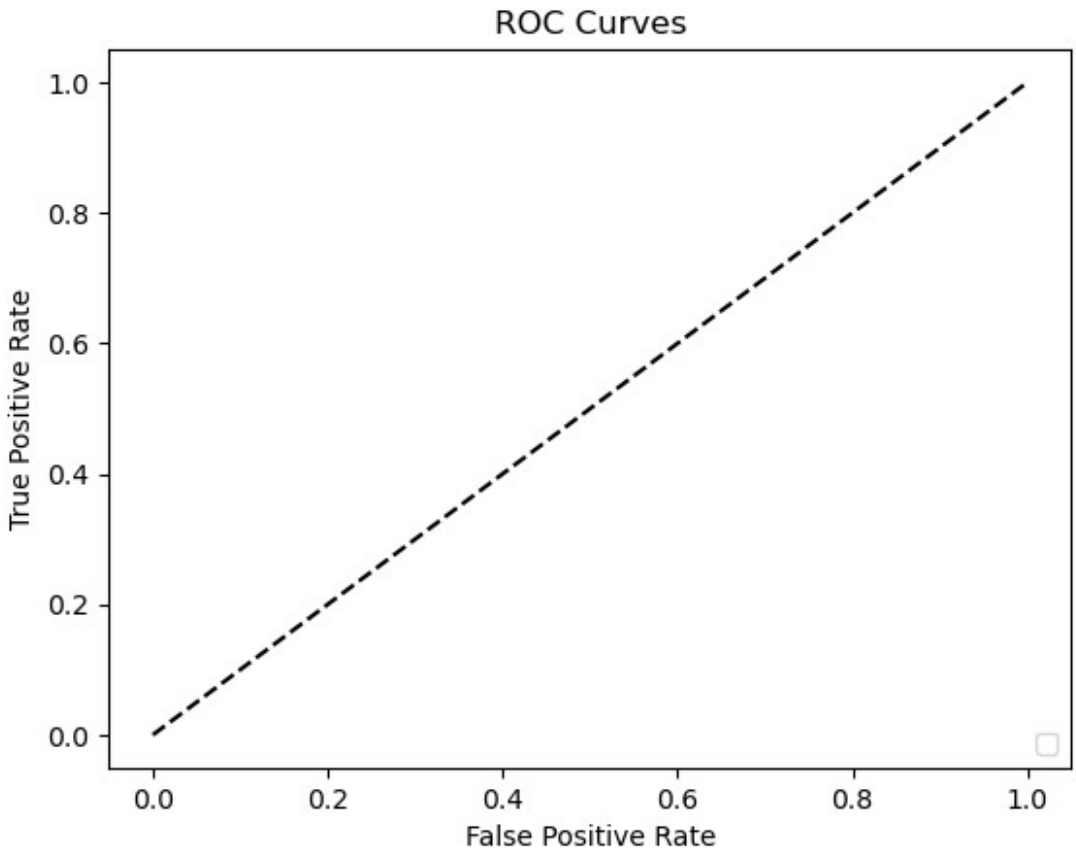



```
# GBM ROC curve
y_pred_proba_gbm = grid_gbm.predict_proba(X_test)[:, 1]
plot_roc_curve(y_test, y_pred_proba_gbm, 'GBM')
```




```
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves')
plt.legend(loc='lower right')
plt.show()
```

```
No artists with labels found to put in legend.  Note that artists
whose label start with an underscore are ignored when legend() is
called with no argument.
```



Model Comparison and Reflection

Based on the evaluation metrics, the best performing model was with the following key observations:

- SVM: SVM showed strong performance, particularly in scenarios where the kernel effectively captured non-linear relationships in the data.
- Random Forest: Random Forest excelled in terms of precision, handling noisy data effectively due to its ensemble approach.
- GBM: Gradient Boosting demonstrated consistent performance across metrics, excelling in its ability to handle complex, imbalanced data.

Impact of Hyperparameter Tuning:

Hyperparameter tuning had a significant effect on the performance of all models. The tuning process refined each model's ability to balance the trade-offs between underfitting and overfitting:

- SVM benefited from an optimal selection of the kernel and regularization parameter, resulting in more precise decision boundaries.
- Random Forest performance improved significantly with a larger number of estimators and deeper trees.

- GBM: Tuning the learning rate and number of boosting stages helped prevent overfitting and improved overall performance.

Conclusion:

In this report, we successfully applied three machine learning models to predict heart disease using health indicators. This achieved the best balance between accuracy, precision, and recall after hyperparameter tuning. The models performed well, each with unique strengths.

The model with the highest values in these metrics, especially AUC-ROC,here both random forest and GBM model has high Area under the curve(AUC-ROC) and is typically considered the best performing models.

The choice of model depend on the specific requirements of the healthcare application, where precision or recall might be prioritized depending on whether false positives or false negatives have a greater impact on patient outcomes.