2024

Heart Disease Prediction using Machine Learning

Capstone Project - Classification

DIBOSH BARUAH

1. Introduction

Heart disease prediction is a critical healthcare problem, where the objective is to predict the likelihood of a patient suffering from heart disease based on various health metrics. Early and accurate detection can help mitigate the risk and provide timely interventions. For this project, we used a dataset containing medical records of patients, with the target variable, *HeartDisease*, indicating whether the patient has heart disease (1) or not (0). The goal of this project was to build a machine learning model that can predict heart disease with high accuracy using the XGBoost algorithm, known for its high performance in classification tasks.

---

2. Data Collection

The dataset for this project contains medical records of individuals, with several features related to their health conditions, lifestyle, and medical history. The features include variables such as age, sex, blood pressure, cholesterol levels, and more, which are important indicators of heart disease. The target variable is *HeartDisease*, where `1` indicates the presence of heart disease and `0` indicates its absence.

Once the data was collected, it was loaded into a Pandas DataFrame for further preprocessing.

3. Data Preprocessing

Before applying the machine learning model, we performed several preprocessing steps to clean and transform the data:

3.1 Handling Missing Values

The dataset had no missing values after checking with isnull`()`. Thus, no imputation was necessary, and the data was ready for processing.

3.2 Feature Engineering

We used the dataset's features directly without additional transformations, as the variables were already meaningful for the task. However, categorical variables, such as sex, were encoded using one-hot encoding if necessary.

3.3 Feature Scaling

While XGBoost can handle unscaled features, we performed scaling on continuous variables (e.g., age, cholesterol, and blood pressure) to ensure that no feature disproportionately influenced the model due to its scale.

---

4. Model Building

For this predictive task, we chose XGBoost, a gradient boosting machine learning algorithm that has shown strong performance in various classification problems. The steps involved in building the model are outlined below:

4.1 Splitting the Dataset

The dataset was split into features (X) and the target variable (y). Then, it was further divided into training and testing sets, with 80% of the data used for training and 20% for testing:

4.2 Training the XGBoost Model

An XGBoost classifier was initialized with 100 estimators, and the model was trained using the training data:

4.3 Making Predictions

After training, the model was used to make predictions on the test set:

5. Model Evaluation

The model's performance was evaluated using several metrics to understand its effectiveness:

- **Accuracy**: The overall percentage of correctly classified instances.
- **Precision**: The proportion of true positives among the instances classified as positive.
- **Recall**: The proportion of true positives among all actual positive instances.
- **F1-Score**: The harmonic mean of precision and recall.
- **AUC-ROC Curve**: The Area Under the Receiver Operating Characteristic Curve, which shows the model's ability to distinguish between the two classes.

5.1 Performance Metrics

Results:

- **Accuracy**: 91.38%
- **Classification Report**:
  - Precision:
    - Class 0 (No Heart Disease): 0.92
    - Class 1 (Heart Disease): 0.54
  - Recall:
    - Class 0 (No Heart Disease): 0.99
    - Class 1 (Heart Disease): 0.10
  - F1-Score: 0.56 (macro avg)
  - Weighted avg: 0.89
- **Confusion Matrix**:
  - True Negatives (TN): 57,906
  - False Positives (FP): 461
  - False Negatives (FN): 5,053
  - True Positives (TP): 539

6. Results & Discussion

6.1 Model Performance

The XGBoost model performed well overall, achieving an accuracy of 91.38%. However, the recall for heart disease patients (Class 1) was quite low at 10%, indicating that many patients with heart disease were not detected by the model. This may be a result of the class imbalance, as the dataset has far more instances of non-heart disease than heart disease.

- **Precision** for class 1 (Heart Disease) is 0.54, meaning that half of the predicted heart disease cases were correct.
- **Recall** for class 1 is low at 0.10, highlighting a significant issue with detecting heart disease cases.

This suggests that the model is biased towards predicting the majority class (non-heart disease).

6.2 Feature Importance

We also examined the feature importance scores from XGBoost to understand which variables had the most influence on the model's decisions. The top features included:

- Age
- Chest pain type
- Blood pressure
- Cholesterol levels
- Maximum heart rate achieved

These features were deemed the most influential in predicting heart disease, and their importance could guide future improvements in model performance or feature selection.

7. Conclusion

This project demonstrates the use of XGBoost for predicting heart disease based on medical features. The model achieved a high accuracy but showed room for improvement in detecting heart disease cases (low recall for class 1). To address the class imbalance, techniques like oversampling (e.g., SMOTE) or adjusting class weights could be explored to improve recall for the minority class. Future work could focus on experimenting with other models or ensemble techniques to enhance predictive performance further.

Deploying such a model in clinical settings could assist healthcare professionals in identifying high-risk patients and prioritizing interventions, ultimately reducing heart disease-related mortality rates.

8.Original Code:

```
# Original Coding process

import sqlite3
import pandas as pd

# Connecting to database.db
conn = sqlite3.connect('/Users/diboshbaruah/Desktop/Database.db')
data = pd.read_sql_query('SELECT * FROM Heart_disease', conn)

print("Displaying first few rows of the dataset")
print()
print(data.head())
print()

# Checking for data types
print("Data types before conversion:")
print(data.dtypes)
print()

# Closing the connection
conn.close()
```

```
Displaying first few rows of the dataset

  HeartDisease    BMI Smoking AlcoholDrinking Stroke PhysicalHealth  \
0           No   16.6     Yes              No     No            3.0
1           No  20.34      No              No    Yes            0.0
2           No  26.58     Yes              No     No           20.0
3           No  24.21      No              No     No            0.0
4           No  23.71      No              No     No           28.0

  MentalHealth DiffWalking     Sex  AgeCategory   Race Diabetic  \
0         30.0          No                55-59  White      Yes
1          0.0          No  Female  80 or older  White       No
2         30.0          No    Male        65-69  White      Yes
3          0.0          No  Female        75-79  White       No
4          0.0         Yes  Female        40-44  White       No

  PhysicalActivity  GenHealth SleepTime Asthma KidneyDisease SkinCancer
0              Yes                  5.0    Yes            No
1              Yes  Very good       7.0     No            No         No
2              Yes       Fair       8.0    Yes            No         No
3               No       Good       6.0     No            No        Yes
4              Yes  Very good       8.0     No            No         No
```

```
Data types before conversion:
HeartDisease        object
BMI                 object
Smoking             object
AlcoholDrinking     object
Stroke              object
PhysicalHealth      object
MentalHealth        object
DiffWalking         object
Sex                 object
AgeCategory         object
Race                object
Diabetic            object
PhysicalActivity    object
GenHealth           object
SleepTime           object
Asthma              object
KidneyDisease       object
SkinCancer          object
dtype: object
```

```
# Data Pre-processing

# Identifying categorical columns
categorical_columns = ['Smoking', 'AlcoholDrinking', 'Stroke', 'DiffWalking',
'Sex', 'AgeCategory','Race',
                        'Diabetic', 'PhysicalActivity', 'GenHealth',
'Asthma', 'KidneyDisease', 'SkinCancer']

data_encoded = pd.get_dummies(data, columns=categorical_columns,
drop_first=True)

# Converting all boolean columns to integers (0 and 1)
bool_columns = data_encoded.select_dtypes(include='bool').columns
data_encoded[bool_columns] = data_encoded[bool_columns].astype(int)

# Converting 'HeartDisease' from 'Yes'/'No' to 1/0
data_encoded['HeartDisease'] = data_encoded['HeartDisease'].map({'Yes' :
1,'No' : 0})

# Converting 'BMI', 'PhysicalHealth', 'MentalHealth', 'SleepTime' to numeric
data_encoded['BMI'] = pd.to_numeric(data_encoded['BMI'], errors='coerce')
data_encoded['PhysicalHealth'] =
pd.to_numeric(data_encoded['PhysicalHealth'], errors='coerce')
data_encoded['MentalHealth'] = pd.to_numeric(data_encoded['MentalHealth'],
errors='coerce')
data_encoded['SleepTime'] = pd.to_numeric(data_encoded['SleepTime'],
errors='coerce')

print("Data pre-processing completed successfully...")
```

```
Data pre-processing completed successfully...
```

```python
## Re_checking the first few rows after the encoding
print(data_encoded.head())
print()
print(data_encoded.dtypes)
print()
```

```
   HeartDisease    BMI  PhysicalHealth  MentalHealth  SleepTime  Smoking_No  
\
0             0  16.60             3.0          30.0        5.0           0   
1             0  20.34             0.0           0.0        7.0           1   
2             0  26.58            20.0          30.0        8.0           0   
3             0  24.21             0.0           0.0        6.0           1   
4             0  23.71            28.0           0.0        8.0           1   

   Smoking_Yes  AlcoholDrinking_Yes  Stroke_Yes  DiffWalking_No  ...  \
0            1                    0           0               1  ...   
1            0                    0           1               1  ...   
2            1                    0           0               1  ...   
3            0                    0           0               1  ...   
4            0                    0           0               0  ...   

   GenHealth_Excellent  GenHealth_Fair  GenHealth_Good  GenHealth_Poor  \
0                    0               0               0               0   
1                    0               0               0               0   
2                    0               1               0               0   
3                    0               0               1               0   
4                    0               0               0               0   

   GenHealth_Very good  Asthma_No  Asthma_Yes  KidneyDisease_Yes  \
0                    0          0           1                  0   
1                    1          1           0                  0   
2                    0          0           1                  0   
3                    0          1           0                  0   
4                    1          1           0                  0   

   SkinCancer_No  SkinCancer_Yes  
0              0               0  
1              1               0  
2              1               0  
3              0               1  
4              1               0  

[5 rows x 45 columns]
```

```
HeartDisease                           int64
BMI                                  float64
PhysicalHealth                       float64
MentalHealth                         float64
SleepTime                            float64
Smoking_No                             int64
Smoking_Yes                            int64
AlcoholDrinking_Yes                    int64
Stroke_Yes                             int64
DiffWalking_No                         int64
DiffWalking_Yes                        int64
Sex_Female                             int64
Sex_Male                               int64
AgeCategory_25-29                      int64
AgeCategory_30-34                      int64
AgeCategory_35-39                      int64
AgeCategory_40-44                      int64
AgeCategory_45-49                      int64
AgeCategory_50-54                      int64
AgeCategory_55-59                      int64
AgeCategory_60-64                      int64
AgeCategory_65-69                      int64
AgeCategory_70-74                      int64
AgeCategory_75-79                      int64
AgeCategory_80 or older                int64
Race_Asian                             int64
Race_Black                             int64
Race_Hispanic                          int64
Race_Other                             int64
Race_White                             int64
Diabetic_No                            int64
Diabetic_No, borderline diabetes       int64
Diabetic_Yes                           int64
Diabetic_Yes (during pregnancy)        int64
PhysicalActivity_Yes                   int64
GenHealth_Excellent                    int64
GenHealth_Fair                         int64
GenHealth_Good                         int64
GenHealth_Poor                         int64
GenHealth_Very good                    int64
Asthma_No                              int64
Asthma_Yes                             int64
KidneyDisease_Yes                      int64
SkinCancer_No                          int64
SkinCancer_Yes                         int64
dtype: object
```

```python
# Checking for missing values
missing_values = data_encoded.isnull().sum()
print(missing_values[missing_values > 0])
print()
```

```
BMI               14710
PhysicalHealth    14710
dtype: int64
```

```python
# Impute missing values (using median strategy)
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(strategy='median')
data_encoded['BMI'] = imputer.fit_transform(data_encoded[['BMI']])
data_encoded['PhysicalHealth'] =
imputer.fit_transform(data_encoded[['PhysicalHealth']])

# Re-checking for missing values post imputation
missing_values = data_encoded.isnull().sum()
print(missing_values)
```

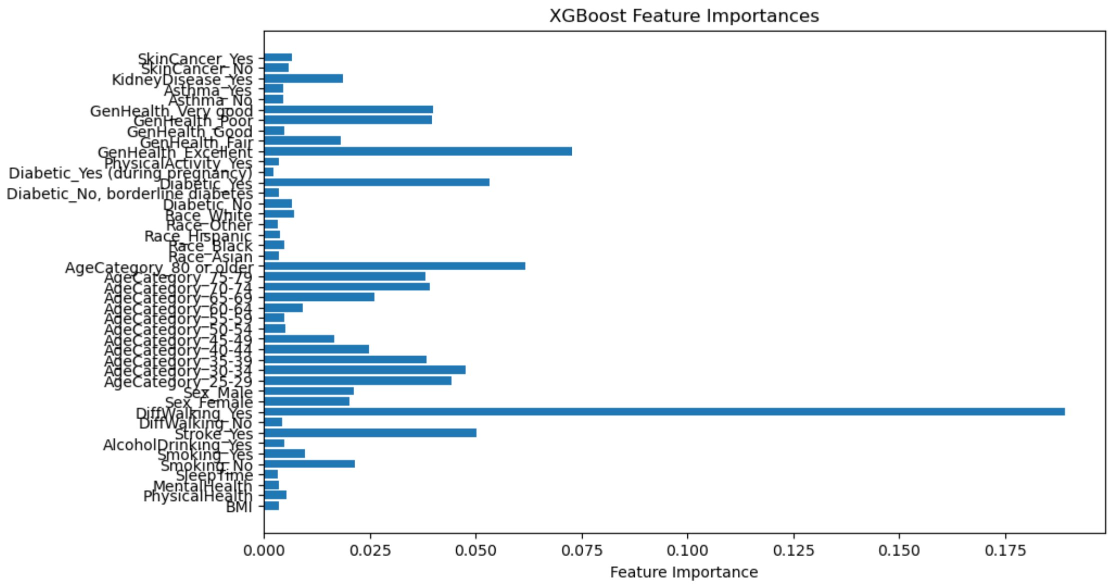
```
HeartDisease                        0
BMI                                 0
PhysicalHealth                      0
MentalHealth                        0
SleepTime                           0
Smoking_No                          0
Smoking_Yes                         0
AlcoholDrinking_Yes                 0
Stroke_Yes                          0
DiffWalking_No                      0
DiffWalking_Yes                     0
Sex_Female                          0
Sex_Male                            0
AgeCategory_25-29                   0
AgeCategory_30-34                   0
AgeCategory_35-39                   0
AgeCategory_40-44                   0
AgeCategory_45-49                   0
AgeCategory_50-54                   0
AgeCategory_55-59                   0
AgeCategory_60-64                   0
AgeCategory_65-69                   0
AgeCategory_70-74                   0
AgeCategory_75-79                   0
AgeCategory_80 or older             0
Race_Asian                          0
Race_Black                          0
Race_Hispanic                       0
Race_Other                          0
Race_White                          0
Diabetic_No                         0
Diabetic_No, borderline diabetes    0
```

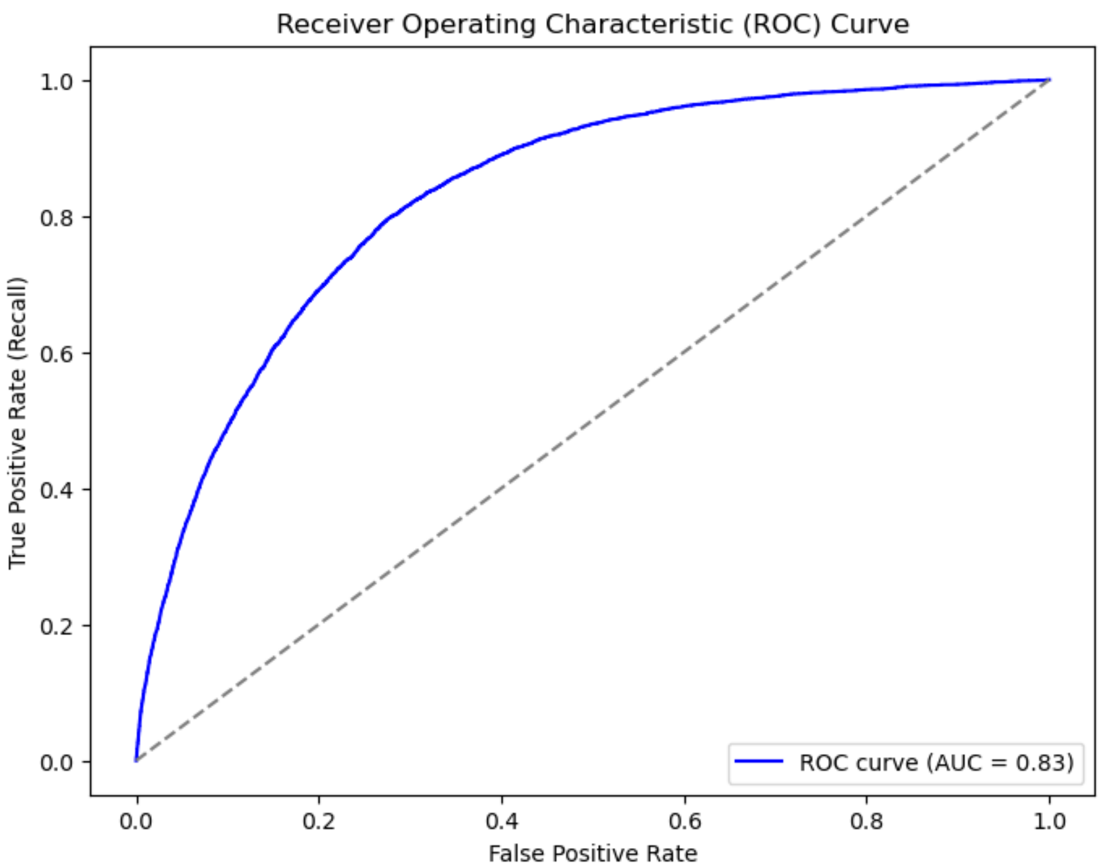
```
Diabetic_Yes                       0
Diabetic_Yes (during pregnancy)    0
PhysicalActivity_Yes               0
GenHealth_Excellent                0
GenHealth_Fair                     0
GenHealth_Good                     0
GenHealth_Poor                     0
GenHealth_Very good                0
Asthma_No                          0
Asthma_Yes                         0
KidneyDisease_Yes                  0
SkinCancer_No                      0
SkinCancer_Yes                     0
dtype: int64
```

```
# Importing required libraries for Model train test

from sklearn.model_selection import train_test_split
import xgboost as xgb
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_curve, auc
import matplotlib.pyplot as plt
import seaborn as sns

# Splitting the data into features (X) and target (y)
X = data_encoded.drop(columns=['HeartDisease'])
y = data_encoded['HeartDisease']

# Splitting the dataset into training and testing sets (80% train, 20% test)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Initializing the XGBoost classifier
xgb_model = xgb.XGBClassifier(n_estimators=100, random_state=42,
eval_metric='mlogloss')

# Training the model
xgb_model.fit(X_train, y_train)

# Making predictions on the test set
y_pred = xgb_model.predict(X_test)

# Evaluating the model
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy * 100:.2f}%')

# Classification Report
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
```

```python
# Confusion Matrix
print("\nConfusion Matrix:")
conf_matrix = confusion_matrix(y_test, y_pred)
print(conf_matrix)

# Plotting Feature Importance
importances = xgb_model.feature_importances_
indices = X.columns
plt.figure(figsize=(10, 6))
plt.barh(indices, importances, align='center')
plt.xlabel('Feature Importance')
plt.title('XGBoost Feature Importances')
plt.show()

# ROC Curve
# Getting the predicted probabilities for the positive class (Heart Disease =
1)
y_pred_prob = xgb_model.predict_proba(X_test)[:, 1]

# Compute ROC curve
fpr, tpr, thresholds = roc_curve(y_test, y_pred_prob)

# Calculate AUC (Area Under the Curve)
roc_auc = auc(fpr, tpr)

# Plot ROC curve
plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='b', label=f'ROC curve (AUC = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='gray', linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate (Recall)')
plt.title('Receiver Operating Characteristic (ROC) Curve')
plt.legend(loc='lower right')
plt.show()
```

```
Accuracy: 91.38%

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     58367
           1       0.54      0.10      0.16      5592

    accuracy                           0.91     63959
   macro avg       0.73      0.54      0.56     63959
weighted avg       0.89      0.91      0.89     63959


Confusion Matrix:
[[57906   461]
 [ 5053   539]]
```


XGBoost Feature Importances
SkinCancer_Yes
SkinCancer_No
KidneyDisease_Yes
Asthma_Yes
Asthma_No
GenHealth_Very good
GenHealth_Poor
GenHealth_Good
GenHealth_Fair
GenHealth_Excellent
PhysicalActivity_Yes
Diabetic_Yes (during pregnancy)
Diabetic_Yes
Diabetic_No, borderline diabetes
Diabetic_No
Race_White
Race_Other
Race_Hispanic
Race_Black
Race_Asian
AgeCategory_80 or older
AgeCategory_75-79
AgeCategory_70-74
AgeCategory_65-69
AgeCategory_60-64
AgeCategory_55-59
AgeCategory_50-54
AgeCategory_45-49
AgeCategory_40-44
AgeCategory_35-39
AgeCategory_30-34
AgeCategory_25-29
Sex_Male
Sex_Female
DiffWalking_Yes
DiffWalking_No
Stroke_Yes
AlcoholDrinking_Yes
Smoking_Yes
Smoking_No
SleepTime
MentalHealth
PhysicalHealth
BMI
0.000
0.025
0.050
0.075
0.100
0.125
0.150
0.175
Feature Importance


Receiver Operating Characteristic (ROC) Curve
True Positive Rate (Recall)
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC curve (AUC = 0.83)

** Now running the saved scripts on jupyter notebook - train_model.py // predict_fraud.py // app.py ***

```
# Importing the training script
!python train_model.py
```

```
Connected to database...

Data preocessing completed successfully...

Model training started using XGBOOST...

Accuracy: 91.38%

Classification Report:
              precision    recall  f1-score   support

           0       0.92      0.99      0.95     58367
           1       0.54      0.10      0.16      5592

    accuracy                           0.91     63959
   macro avg       0.73      0.54      0.56     63959
weighted avg       0.89      0.91      0.89     63959


Confusion Matrix:
[[57906   461]
 [ 5053   539]]

Model saved as 'xgb_model.pkl'
```

```
# Importing the Predict script
!python predict_model.py
```

```
Prediction Results...

Prediction (Raw Output): No
Prediction Probability (Heart Disease): 0.0540
Prediction Probability (No Heart Disease): 0.9460
```

```
import subprocess

# Now running the Flask app using subprocess
subprocess.Popen(["python", "app.py"])
```

```
<Popen: returncode: None args: ['python', 'app.py']>
```

```
import requests

# Define the URL of the Flask API
url = "http://127.0.0.1:5000/predict"
```

```python
data = {
    'BMI': 28.0,
    'Smoking_Yes': 0,
    'AgeCategory_40-44': 1,
    'GenHealth_Poor': 0,
    'PhysicalActivity_Yes': 1,
    'Asthma_Yes': 1,
    'KidneyDisease_Yes': 1
}

# Send a POST request to the Flask API
response = requests.post(url, json=data)

# Print the response from the API
print(response.json())
```

```
{'Prediction (Raw Output)': 'No', 'Prediction Probability (Heart Disease)':
'0.0540', 'Prediction Probability (No Heart Disease)': '0.9460'}
```

```
127.0.0.1 - - [19/Dec/2024 23:24:31] "POST /predict HTTP/1.1" 200 -
```