PROJECT REPORT

CARDIOVASCULAR DISEASE (CVD) DETECTION

SUBMITTED BY

GROUP 12

HIRA AKRAM

ZIDIO DATA SCIENCE INTERNSHIP

Table of Content

1. Introduction

The objective of this project is to create a machine learning model that can predict the probability of cardiovascular disease (CVD) by utilizing relevant health factors. Early detection can facilitate timely treatments and enhance patient outcomes for CVD, a serious health concern. We can evaluate a person's health data, including age, blood pressure, cholesterol levels, and other factors, to estimate their risk of getting CVD by utilizing machine learning algorithms. The goal of this research is to develop an accurate and reliable instrument for estimating the risk of CVD.

2. Required Hardware and Software

To undertake this project, the following hardware and software are required:

- Hardware:

  - Computer with sufficient processing power and memory (RAM (8GB)).
  - Storage capacity to store the dataset and trained models.

- Software:

  - Python programming language and relevant libraries/frameworks, such as scikit-learn, Pandas, and NumPy.
  - Integrated Development Environment : Jupyter Notebook.
  - Data visualization tools, such as Matplotlib or Seaborn.

3. Data Collection and Preprocessing

The data for this project has been provided by the team. As part of the preprocessing steps, we need to handle missing values in the dataset. Here are a few approaches which were used to fix the missing values:

- Mean/Average Imputation:

  Missing values can be filled with the mean or average value of the available data in that column. This approach assumes that the missing values are missing at random and that the mean value is representative of the overall distribution.

- Median Imputation:

  Similar to mean imputation, for numerical columns, missing values can be filled with the median value of the available data. The median is less sensitive to outliers compared to the mean and can be a better choice when the data is skewed.

- Forward-Fill (ffill) or Backward-Fill (bfill):

  Missing values can be filled by propagating the last observed value forward (ffill) or the next observed value backward (bfill). This approach assumes that the missing values maintain a consistent pattern in relation to the neighboring data points.

4. Machine Learning Model Development

Developing the machine learning model consists of following steps:

- Feature and Label Extraction:

  The features are extracted from the DataFrame by dropping the 'target' column and assigning the remaining columns to the variable 'X'. The target variable is assigned to the variable 'y'.

- Train-Test Split:

  The data is split into training and testing sets using the 'train_test_split' function from the 'sklearn.model_selection' module. The training set is used to train the model, while the testing set is used to evaluate its performance. The testing set size is set to 20% of the data, and a random state of 42 is used for reproducibility.

- Feature Standardization:

  The features in the training and testing sets, represented by 'X_train' and 'X_test', respectively, are standardized using the 'StandardScaler' from the 'sklearn.preprocessing' module. Standardization ensures that the features have zero mean and unit variance, which can improve the performance of certain machine learning algorithms.

- Random Forest Classifier:

  The Random Forest Classifier model is defined using the 'RandomForestClassifier' from the 'sklearn.ensemble' module. The 'random_state' parameter is set to 42 for reproducibility.

- Model Training and Prediction:

  The Random Forest Classifier is trained on the standardized training data using the 'fit' method. Then, the trained model is used to predict the target values for the standardized testing data using the 'predict' method, resulting in 'y_pred_rfc'.

- Model Evaluation:

  Several metrics are computed to evaluate the performance of the Random Forest Classifier. The 'accuracy_score', 'confusion_matrix', and 'classification_report' functions from the 'sklearn.metrics' module are used. The accuracy, confusion matrix, and classification report for the model are calculated and stored in 'accuracy_rfc', 'conf_matrix_rfc', and 'classification_rep_rfc', respectively.

5. Code

```
import pandas as pd
import numpy as np
from matplotlib import pyplot as plt
%matplotlib inline
import matplotlib
matplotlib.rcParams["figure.figsize"] = (20,10)

# Data Load: Load CVD data into a dataframe
df = pd.read_csv('Heart_data.csv')

# Display the first few rows of the dataframe
df.head()

# Display the shape of the dataframe
df.shape

# Display the column names
df.columns

# Display the data types of each column
df.dtypes

# Display basic statistics for numerical columns
df.describe()

# Display concise summary of the dataframe
df.info()

# Data Cleaning: Handle NA values
# Check for missing values in each column
df.isnull().sum()

# Remove rows with missing values in the 'sex' column
df.dropna(subset=['sex'], inplace=True)

# Impute missing values in 'age', 'chol', and 'oldpeak' columns with mean values
```

```
df['age'] = df['age'].fillna(df.age.mean())
df['chol'] = df['chol'].fillna(df.chol.mean())
df['oldpeak'] = df['oldpeak'].fillna(df.oldpeak.mean())

# Impute missing values in 'cp' and 'slope' columns using forward fill method
df['cp'].fillna(method='ffill', inplace=True)
df['slope'].fillna(method='ffill', inplace=True)

# Impute missing values in 'fbs' and 'exang' columns with median values
df['fbs'] = df['fbs'].fillna(df.fbs.median())
df['exang'] = df['exang'].fillna(df.exang.median())

# Drop rows with any remaining missing values
df = df.dropna()

# Check for missing values after cleaning
df.isnull().sum()

# Check unique values in 'target' column
df['target'].unique()

# Plot bar chart for distribution of target variable
df['target'].value_counts().plot.bar()

# Combine classes 2, 3, and 4 in 'target' column into a single class
df['target'] = df['target'].replace([2, 3, 4], 1)

# Check the shape of the dataframe after data cleaning
df.shape

# Build a Model
# Step 1: Extract features and labels
X = df2.drop(columns=['target'], axis=1)
y = df2['target']

# Step 2: Split the data into training and testing sets
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, random_state=42)

# Step 3: Standardize the features
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# Step 4: Define RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier(random_state=42)

# Step 5: Train and evaluate the RandomForestClassifier
```

```
rfc.fit(X_train, y_train)
y_pred_rfc = rfc.predict(X_test)

# Evaluate the RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
accuracy_rfc = accuracy_score(y_test, y_pred_rfc)
conf_matrix_rfc = confusion_matrix(y_test, y_pred_rfc)
classification_rep_rfc = classification_report(y_test, y_pred_rfc)

# Display results for RandomForestClassifier
print("RandomForestClassifier Results:")
print(f'Model Accuracy: {accuracy_rfc}')
print(f'Confusion Matrix:\n{conf_matrix_rfc}')
print(f'Classification Report:\n{classification_rep_rfc}')

# Use K Fold cross validation to measure accuracy of our model
from sklearn.model_selection import cross_val_score
cv_scores = cross_val_score(rfc, X, y, cv=5, scoring='accuracy')

# Display the cross-validation scores
print("\nCross-Validation Scores:", cv_scores)

# Calculate and display the average accuracy
average_accuracy = np.mean(cv_scores)
print("Average Accuracy:", average_accuracy)

# Find best model using GridSearchCV
from sklearn.model_selection import GridSearchCV
def find_best_model_using_gridsearchcv(X, y):
    algos = {
        'random_forest': {
            'model': RandomForestClassifier(),
            'params': {
                'n_estimators': [50, 100, 150],
                'max_depth': [None, 10, 20],
                'min_samples_split': [2, 5, 10],
            }
        },
        'k_neighbors': {
            'model': KNeighborsClassifier(),
            'params': {
                'n_neighbors': [3, 5, 7],
                'weights': ['uniform', 'distance']
            }
        }
    }

    scores = []
    cv = ShuffleSplit(n_splits=5, test_size=0.2, random_state=0)
```

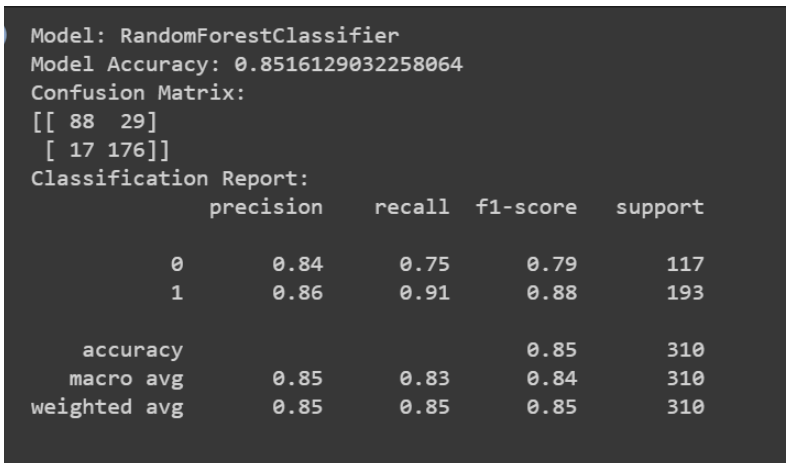
```
    for algo_name, config in algos.items():
        gs = GridSearchCV(config['model'], config['params'], cv=cv, scoring='accuracy',
return_train_score=False)
        gs.fit(X, y)
        scores.append({
            'model': algo_name,
            'best_score': gs.best_score_,
            'best_params': gs.best_params_
        })

    return pd.DataFrame(scores, columns=['model', 'best_score', 'best_params'])

# Call the function and store the results
results_df = find_best_model_using_gridsearchcv(X, y)
print(results_df)

# Export the tested model to a pickle file
import pickle
with open('cvd_model.pickle', 'wb') as f:
    pickle.dump(results_df, f)
```

6. Project Output Screenshots



```
Model: RandomForestClassifier
Model Accuracy: 0.8516129032258064
Confusion Matrix:
[[ 88  29]
 [ 17 176]]
Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.75      0.79       117
           1       0.86      0.91      0.88       193

    accuracy                           0.85       310
   macro avg       0.85      0.83      0.84       310
weighted avg       0.85      0.85      0.85       310
```

7. Conclusion

In conclusion, the CVD Prediction project aims to develop a machine learning model for estimating the likelihood of cardiovascular disease based on relevant health parameters. The project involves data collection, preprocessing, model development, training, and evaluation. Successful completion of this project can contribute to personalized healthcare and improve patient outcomes.

8. References

- https://www.kaggle.com/datasets
- https://chat.openai.com/
- https://github.com/Hira-da/Data-Science