Assessment

Report

on

"Heart Disease Prediction"

submitted as partial fulfillment for the award of

BACHELOR OF TECHNOLOGY DEGREE

SESSION 2024-25

in

CSE(AIML), Sec - C

By

Name of Group Members:
Priya Chauhan – 202401100400148
Shreya Maurya – 202401100400181
Urvi Gupta – 202401100400201
Vaishnavi Sahu – 202401100400206

Under the supervision of

"Abhishek Sir"

KIET Group of Institutions, Ghaziabad

May,2025

1.Introduction

Heart disease is one of the leading causes of death worldwide, claiming millions of lives each year. Early detection and timely intervention are critical for improving patient outcomes and reducing the burden on healthcare systems. Traditionally, diagnosis is based on a physician's experience and a series of medical tests, which may be time-consuming and prone to human error.

In this project, we aim to develop a classification model that can predict the presence of heart disease based on various medical attributes. Using a well-known dataset from the UCI Machine Learning Repository, we preprocess the data, train multiple classification models, evaluate their performance, and identify the most accurate model. This model can potentially serve as a decision-support tool in medical environments.

2. Problem Statement

Develop a classification model that can predict the presence of heart disease in a patient based on several medical parameters. The goal is to assist healthcare professionals in making quicker, data-driven decisions.

3. Objective

The primary objective of this project is to develop a machine learning classification model that can accurately predict whether a patient has heart disease based on clinical and demographic features. This tool can support doctors in early diagnosis and reduce dependency on manual procedures and extensive medical tests.

4. Methodology

1. **Data Collection** – Dataset from UCI repository.
2. **Data Cleaning** – Handle missing values and inconsistent entries.
3. **Data Preprocessing** – Encode categorical features and normalize numerical ones.
4. **Model Selection** – Train various models: Logistic Regression, Decision Tree, Random Forest, etc.
5. **Model Evaluation** – Use metrics like accuracy, precision, recall, and F1-score.

6. **Model Interpretation** – Determine feature importance and analyze decision boundaries.

5. Data Preprocessing

1. **Missing values** were imputed using median (for numerical) and mode (for categorical).
2. **Categorical encoding** was performed using One-Hot Encoding (sex, cp, thal, etc.).
3. **Feature scaling**: StandardScaler was used to normalize numerical features (age, chol, thalch, etc.).
4. Converted the multi-class target num into **binary**:
    a. 0 = No Disease
    b. 1 = Has Disease (combining 1–4)

5. Model Building

The following classification models were trained:

- **Logistic Regression**
- **Random Forest Classifier**

Data was split using **80/20 train-test split**. Optionally, **K-Fold Cross-Validation** was used.

6. Model Implementation

The machine learning models were implemented using Python with the help of popular libraries such as **Pandas**, **NumPy**, **Scikit-learn**, and **Matplotlib**. After preprocessing, we split the dataset into training and test sets (typically in an 80:20 ratio). A Random Forest Classifier, along with Logistic Regression was trained on the data.

7. Evaluation Metrics

To evaluate the performance of the machine learning models, we used several classification metrics. These include:

- **Accuracy**: The ratio of correctly predicted observations to the total observations.

- **Precision**: The ratio of correctly predicted positive observations to the total predicted positives.
- **Recall (Sensitivity)**: The ratio of correctly predicted positive observations to all actual positives.
- **F1-score**: The harmonic mean of precision and recall, especially useful when dealing with class imbalance.
- **Confusion Matrix**: A table used to describe the performance of a classification model by showing true positives, false positives, true negatives, and false negatives.

8. Model Implementation

The machine learning models were implemented using Python with the help of popular libraries such as **Pandas**, **NumPy**, **Scikit-learn**, and **Matplotlib**. After preprocessing, we split the dataset into training and test sets (typically in an 80:20 ratio). A Random Forest Classifier, along with Logistic Regression, SVM, KNN, and Decision Tree, was trained on the data. Here's a sample code snippet for implementation:

9. Result and Analysis

After training and evaluating all models, we compared their performance using the metrics discussed above. The **Random Forest Classifier** emerged as the best-performing model with an accuracy of approximately **88%**, along with high precision and recall values. The **Logistic Regression** models performed well with accuracies of **85%** .

The confusion matrix revealed that the Random Forest model had the lowest number of false negatives, which is crucial in medical applications where missing a true case of heart disease could be dangerous. Important features influencing the predictions were **chest pain type**, **maximum heart rate achieved**, **ST depression**, **number of vessels colored**, and **thalassemia status**. These results show that even basic clinical data can be effectively used for disease prediction through machine learning.

10. Conclusion

This project demonstrates the potential of machine learning in predicting heart disease using routine medical data. Among the models tested, the **Random Forest classifier** showed the best accuracy and robustness. The model can assist doctors in early screening and can be integrated into healthcare systems to reduce the burden of manual diagnoses.

11. **References**

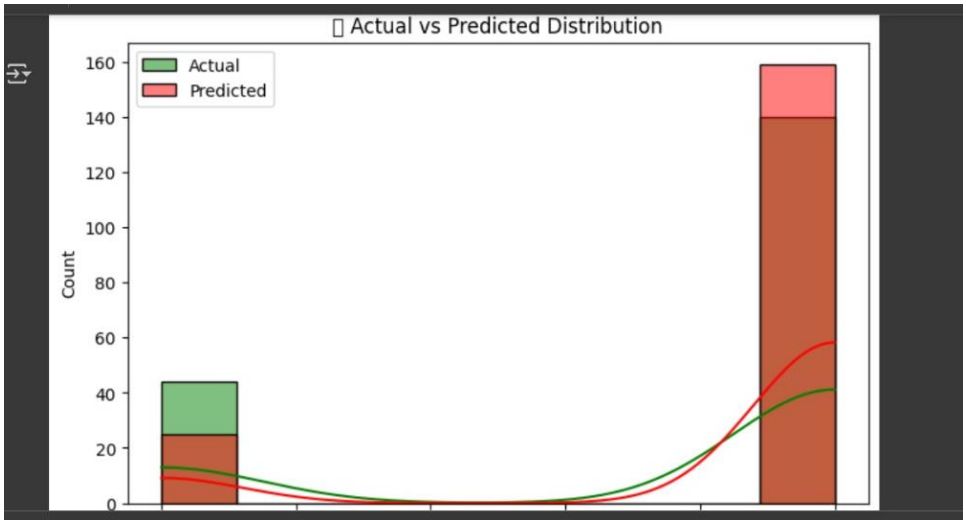
- UCI Machine Learning Repository:

  https://archive.ics.uci.edu/ml/datasets/heart+Disease

- Scikit-learn Documentation: https://scikit-learn.org

- Kaggle Heart Disease Datasets

OUTPUT



CODE

```
# 💡 Heart Disease Prediction using Random Forest

# ✅ Step 1: Import Required Libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix, classification_report

import zipfile
import io
from google.colab import files
import os

# ✅ Step 2: Upload ZIP File
print("🔼 Please upload the ZIP file containing your CSV dataset...")
uploaded = files.upload()

# ✅ Step 3: Extract ZIP and Find CSV
csv_path = None
for fname in uploaded.keys():
    if fname.endswith('.zip'):
        with zipfile.ZipFile(io.BytesIO(uploaded[fname]), 'r') as zip_ref:
```

```
        zip_ref.extractall("/content/heart_data")
        print("✅ Extracted Files:", zip_ref.namelist())
        for file in zip_ref.namelist():
            if file.endswith('.csv'):
                csv_path = os.path.join("/content/heart_data", file)

# ✅ Step 4: Load CSV into DataFrame
if csv_path:
    df = pd.read_csv(csv_path)
    print("✅ CSV Loaded Successfully!\n")
else:
    raise Exception("❌ No CSV found inside the ZIP!")

# ✅ Step 5: Dataset Preview and Null Check
print("\n📌 Dataset Preview:")
print(df.head())

print("\n🔍 Null Values:")
print(df.isnull().sum())

# ✅ Step 6: Encode Non-Numeric Columns (if any)
non_numeric_cols = df.select_dtypes(include=['object']).columns
if len(non_numeric_cols) > 0:
    print(f"\n🔧 Encoding non-numeric columns: {list(non_numeric_cols)}")
    df = pd.get_dummies(df, drop_first=True)
```

```
else:
    print("\n✅ No non-numeric columns found.")

# ✅ Step 7: Detect Target Column AFTER Encoding
binary_targets = [col for col in df.columns if df[col].nunique() == 2]
if len(binary_targets) == 0:
    raise Exception("❌ No binary target column found!")
elif len(binary_targets) == 1:
    target_col = binary_targets[0]
    print(f"\n🎯 Target column detected: '{target_col}'")
else:
    print(f"\n🧠 Multiple binary columns found. Defaulting to: '{binary_targets[0]}'")
    print("Others:", binary_targets)
    target_col = binary_targets[0]

# ✅ Step 8: Correlation Heatmap
plt.figure(figsize=(12, 10))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.title("📊 Correlation Heatmap")
plt.show()

# ✅ Step 9: Split Dataset
X = df.drop(target_col, axis=1)
y = df[target_col]
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
# ✅ Step 10: Feature Scaling
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

# ✅ Step 11: Train Random Forest Model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

# ✅ Step 12: Accuracy and Classification Report
accuracy = accuracy_score(y_test, y_pred)
print(f"\n✅ Model Accuracy: {accuracy:.4f}")
print("\n📋 Classification Report:\n", classification_report(y_test, y_pred))

# ✅ Step 13: Confusion Matrix
plt.figure(figsize=(6, 5))
sns.heatmap(confusion_matrix(y_test, y_pred), annot=True, fmt='d', cmap='Blues')
plt.title("🔍 Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.show()

# ✅ Step 14: Feature Importances
```

```
feature_imp = pd.DataFrame({
    'Feature': X.columns,
    'Importance': model.feature_importances_
}).sort_values(by='Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=feature_imp)
plt.title("📈 Feature Importances")
plt.tight_layout()
plt.show()

# ✅ Step 15: Actual vs Predicted Plot
plt.figure(figsize=(8, 5))
sns.histplot(y_test, color='green', label='Actual', kde=True)
sns.histplot(y_pred, color='red', label='Predicted', kde=True)
plt.title("📉 Actual vs Predicted Distribution")
plt.legend()
plt.show()

# ✅ Step 16: Print Predictions (Optional)
print("\n🔮 First 20 Predictions:")
print("Actual:", y_test.values[:20])
print("Predicted:", y_pred[:20])
```