Heart Disease Prediction Model

Task-02

June 10, 2025

Muhammad Furqan Qureshi
DHC: 3634

Contents

Introduction 2

Objective: 2

Dataset: 2

Data Loading and Cleaning 2

Step 1: Loading the Data 2

Step 2: Checking for Missing Values 2

Step 3: Removing Duplicates 2

Step 4: Handling Irrelevant Features 2

Exploratory Data Analysis (EDA) 3

Step 1: Descriptive Statistics 3

Step 2: Distribution of Target Variable 3

Step 3: Feature Relationships 3

Step 4: Correlation Matrix 3

Data Preprocessing 3

Step 1: Separating Features and Target 3

Step 2: Train-Test Split 4

Step 3: Feature Scaling 4

Model Building 4

Logistic Regression Model 4

Decision Tree Model 4

Model Evaluation 4

Accuracy Score 4

Confusion Matrix 5

ROC Curve and AUC 5

Feature Importance Analysis 5

Conclusion 5

Key Insights 6

References 6

Introduction

Objective:

I built a model to predict whether a person is at risk of heart disease based on various health-related features using machine learning algorithms. I used the Heart Disease UCI Dataset, which contains data about individuals' age, blood pressure, cholesterol levels, etc.

Dataset:

The dataset is publicly available on Kaggle and contains several health metrics such as age, cholesterol, resting blood pressure, and others. The target variable in this dataset indicates whether a person has heart disease (1 for positive, 0 for negative).

Data Loading and Cleaning

Step 1: Loading the Data

I loaded the dataset using pandas, a powerful data manipulation library in Python. I used the following code to load the data:

```
df = pd.read_csv('heart_disease_data.csv')
```

I then displayed the first few rows of the dataset to understand its structure:

```
print(df.head())
```

Step 2: Checking for Missing Values

I checked the dataset for any missing values, as missing data can cause issues in model training. I used the isnull() function to check for missing data

```
df.isnull().sum()
```

If any missing values were found, I filled them with the median of each column

```
df.fillna(df.median(), inplace=True)
```

Step 3: Removing Duplicates

Next, I checked for duplicates in the dataset using the duplicated() function. I removed any duplicate rows if they existed

```
df.drop_duplicates(inplace=True)
```

Step 4: Handling Irrelevant Features

After examining the dataset, I identified that some columns were not useful for prediction (such as an ID column). I removed these columns to simplify the model:

```
df.drop(['column_name'], axis=1, inplace=True)
```

Exploratory Data Analysis (EDA)

Step 1: Descriptive Statistics

I used the `describe()` function to generate a summary of statistics for numerical columns. This helped me understand the range, mean, and distribution of each feature
`df.describe()`

Step 2: Distribution of Target Variable

To understand the distribution of the target variable (whether or not someone has heart disease), I plotted the count of each class

```
sns.countplot(x='target', data=df)
plt.title('Heart Disease Distribution')
plt.show()
```

Step 3: Feature Relationships

I visualized relationships between key features (e.g., age, cholesterol, resting blood pressure) and the target variable using pair plots. This helped me understand how the features relate to the target:

```
sns.pairplot(df, hue='target', vars=['age', 'trestbps',
'chol', 'thalachh'])
plt.show()
```

Step 4: Correlation Matrix

I generated a heatmap to visualize the correlation between the features. This allowed me to identify highly correlated features that may be important for the model:

```
sns.heatmap(df.corr(), annot=True, cmap='coolwarm')
plt.show()
```

Data Preprocessing

Step 1: Separating Features and Target

I separated the features (input variables) and the target variable (output). The target variable is `target`, while the other columns are the features

```
X = df.drop('target', axis=1)
y = df['target']
```

Step 2: Train-Test Split

I split the dataset into training and testing sets using an 80-20 split. This helps me train the model on one set of data and evaluate it on a separate set:

```
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
```

Step 3: Feature Scaling

Since models like Logistic Regression are sensitive to the scale of the data, I standardized the features using `StandardScaler`:

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

Model Building

Logistic Regression Model

I first trained a Logistic Regression model, a simple binary classification model, to predict whether someone has heart disease. I used the following code to initialize and train the model

```
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
```

I then made predictions on the test data

```
y_pred_log_reg = log_reg.predict(X_test_scaled)
```

Decision Tree Model

Next, I trained a Decision Tree classifier, which is more interpretable and can show the importance of different features in making predictions

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, y_train)
```

I made predictions using the Decision Tree model:

```
y_pred_tree = decision_tree.predict(X_test)
```

Model Evaluation

Accuracy Score

I evaluated both models using the accuracy score, which measures the proportion of correct predictions

```
accuracy_log_reg = accuracy_score(y_test, y_pred_log_reg)
accuracy_tree = accuracy_score(y_test, y_pred_tree)
```

Confusion Matrix

I used confusion matrices to evaluate the models' performance in terms of false positives, false negatives, and overall classification performance

```
conf_matrix_log_reg = confusion_matrix(y_test, y_pred_log_reg)
conf_matrix_tree = confusion_matrix(y_test, y_pred_tree)
```

ROC Curve and AUC

I evaluated the models' ability to distinguish between the two classes using the ROC curve and AUC score. I plotted the ROC curves for both models

```
fpr_log_reg, tpr_log_reg, _ = roc_curve(y_test,
log_reg.predict_proba(X_test_scaled)[:, 1])
roc_auc_log_reg = roc_auc_score(y_test, y_pred_log_reg)

fpr_tree, tpr_tree, _ = roc_curve(y_test,
decision_tree.predict_proba(X_test)[:, 1])
roc_auc_tree = roc_auc_score(y_test, y_pred_tree)
```

Feature Importance Analysis

Since the Decision Tree model provides insights into which features are important, I visualized the feature importance using a bar plot. This helps identify the most important factors in predicting heart disease

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

sns.barplot(x='Importance', y='Feature', data=importance_df)
plt.title('Feature Importance (Decision Tree)')
plt.show()
```

Conclusion

In conclusion, I compared the performance of Logistic Regression and Decision Tree models. Both models provided useful insights into predicting heart disease, but the Decision Tree model also helped identify the most important features that affect the prediction. The features like age, cholesterol levels, and resting blood pressure were the most significant in predicting heart disease.

Key Insights

- Logistic Regression provided a solid baseline, but Decision Trees were more interpretable and showed clear feature importance.
- The most important features were `age`, `cholesterol`, and `trestbps` (resting blood pressure).

References

- **Dataset**: Heart Disease UCI Dataset (available on Kaggle)

- **Libraries Used**:

  - **Pandas:** for data manipulation
  - **Seaborn:** for visualizations
  - **Scikit-learn:** for model building and evaluation