IML_Heart_Disease_Prediction

May 6, 2024

Tyler Floyd, Michael Daniel, Vaishnav Nimmagadda

Relevant Modules

For more information on linear regression, gradiant descent, and the cost function, visit the assignment in the main branch of the github repository in the following code.

[2]:
```
import numpy as np #Numerical Operations
import pandas as pd #Data Manipulation
import matplotlib.pyplot as plt #Plotting
import warnings #Warning Handling Module
import seaborn as sns #Heatmap for Confusion Matrix
from sklearn import metrics #Model Metrics
from sklearn.metrics import confusion_matrix, mean_squared_error #Confusion␣
↪Matrix and MSE
from sklearn.preprocessing import StandardScaler, MinMaxScaler #Standardization␣
↪and Min-Max Normalization
from sklearn.linear_model import LogisticRegression, LinearRegression␣
↪#Regression Training
from sklearn.naive_bayes import GaussianNB #Naive Bayes Classifier
from sklearn.decomposition import PCA #Principal Component Analysis
from sklearn.exceptions import DataConversionWarning #Data Warning
from sklearn.feature_selection import SelectKBest, f_regression #Feature␣
↪Selection
from sklearn.tree import DecisionTreeClassifier #Regression model

warnings.filterwarnings("ignore") #Mute warnings in output

test = pd.read_csv('https://raw.githubusercontent.com/VoluSign/UNCC/main/
↪test_heart.csv')#*
train = pd.read_csv('https://raw.githubusercontent.com/VoluSign/UNCC/main/
↪train_heart.csv')#*
```

Functions

[3]:
```
#Standardization
def Standardize(X_test,X_train):
  sc = StandardScaler()
  X_train = sc.fit_transform(X_train)
  X_test = sc.transform(X_test)
  return X_test,X_train

#Normalization
def Normalize(X_test,X_train):
  scaler = MinMaxScaler()
  X_train = scaler.fit_transform(X_train)
  X_test = scaler.transform(X_test)
  return X_test,X_train

def disMetrics(y_test, y_pred):
  #Display Model Metrics
  print("Accuracy:", metrics.accuracy_score(y_test,y_pred))
  print("Precision:", metrics.precision_score(y_test,y_pred))
  print("Recall:", metrics.recall_score(y_test,y_pred))
  print("F1 score:", metrics.f1_score(y_pred, y_test, average="weighted"))

def conMatrix(y_test, y_pred):
  #Create Confusion Matrix
  class_names = [0,1]
  fig, ax = plt.subplots()
  tick_marks = np.arange(len(class_names))
  plt.xticks(tick_marks, class_names)
  plt.yticks(tick_marks, class_names)
  #Create heatmap
  sns.heatmap(pd.DataFrame(confusion_matrix(y_test, y_pred)), annot = True,␣
 ↪fmt='g')
  #Plot Confusion Matrix
  plt.tight_layout()
  plt.title('Confusion Matrix')
  plt.ylabel('Actual')
  plt.xlabel('Predicted')

def reset(test_X, train_X, test_target, train_target):
  #Create dataframe for data
  X__test = pd.DataFrame(test_X).drop(columns=['target'])
  Y__test = pd.DataFrame(test_target)
  X__train = pd.DataFrame(train).drop(columns=['target'])
  Y__train = pd.DataFrame(train_target)
  X_train, X_test, y_train, y_test = X__train, X__test, Y__train, Y__test
  return X_train, X_test, y_train, y_test
```

Feature Selection

[4]:
```
#Split the data into training and testing sets
X_train, X_test, y_train, y_test = reset(test, train, test.target, train.target)
# Configure to select all features
fs = SelectKBest(score_func=f_regression, k='all')

# Learn relationship from training data
fs.fit(X_train, y_train)

#Get column names
column_names = X_train.columns

feature_weights = {}
for i in range(len(fs.scores_)):
    feature_weights[column_names[i]] = fs.scores_[i]

feature_weights = pd.DataFrame.from_dict(feature_weights, orient='index',␣
 ↪columns=['Feature Weight']).round(2).transpose()

feature_weights.head()
```

[4]:
```
                  age    sex      cp  trestbps   chol   fbs  restecg  thalach  \
Feature Weight  56.79  86.69  238.56     20.09  10.33  1.74    18.84    222.8

                 exang  oldpeak   slope      ca   thal
Feature Weight  242.88   243.45  138.68  174.88  131.8
```

Linear Regression

[5]:
```
# Linear regression model based on heavily weighted features
X_train, X_test, y_train, y_test = reset(test, train, test.target, train.target)
X_train = X_train.drop(columns=['chol','trestbps','age','sex', 'cp', 'fbs',␣
 ↪'restecg', 'exang'])
X_test = X_test.drop(columns=['chol','trestbps','age','sex', 'cp', 'fbs',␣
 ↪'restecg', 'exang'])
X_test, X_train = Standardize(X_test, X_train)  # Standardizes features
model = LinearRegression()  # Linear Regression Model
model.fit(X_train, y_train)  # Model Training
y_pred = model.predict(X_test)  # Predict on test set

print("Mean Squared Error:", mean_squared_error(y_test, y_pred))
```

```
Mean Squared Error: 0.15014826710024817
```

Chol has small effect, can be turned into classification problem using sigmoid function.

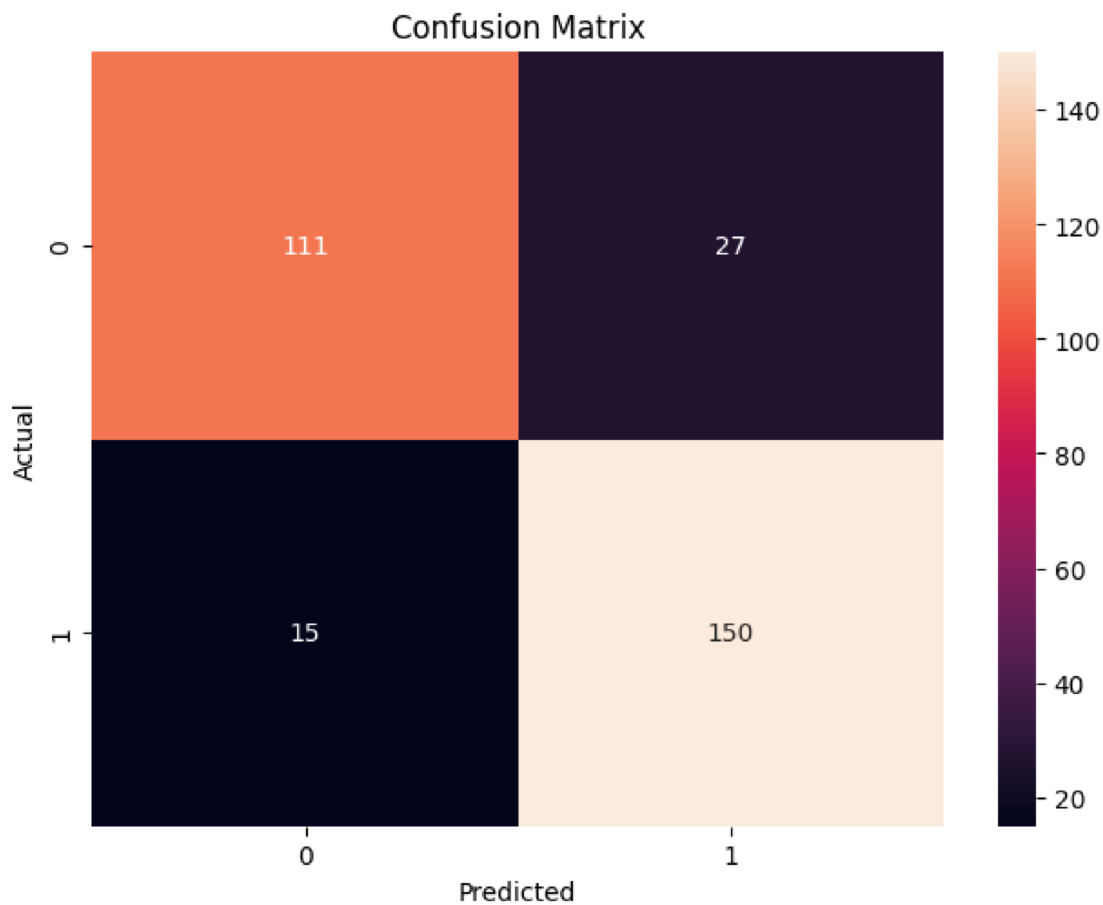
Logistic Regression

[6]:
```
#Logistic Regression Model
X_train, X_test, y_train, y_test = reset(test, train, test.target, train.target)
X_test,X_train = Standardize(X_test,X_train) #Standardizes features
model = LogisticRegression(random_state=0) #Logistic Regression Model
model.fit(X_train, y_train) #Model Training
y_pred = model.predict(X_test) #Predict on test set

disMetrics(y_test, y_pred)
```

```
Accuracy: 0.8613861386138614
Precision: 0.847457627118644
Recall: 0.9090909090909091
F1 score: 0.862104631515783
```

[7]:
```
#Logistic Regression Model With Reularization
model = LogisticRegression(penalty='l2', C=1, random_state=0) #Logistic␣
 ↪Regression Model With Reularization (lambda = .03)
model.fit(X_train, y_train) #Model Training
y_pred = model.predict(X_test) #Predict on test set

disMetrics(y_test, y_pred)
conMatrix(y_test, y_pred)
```

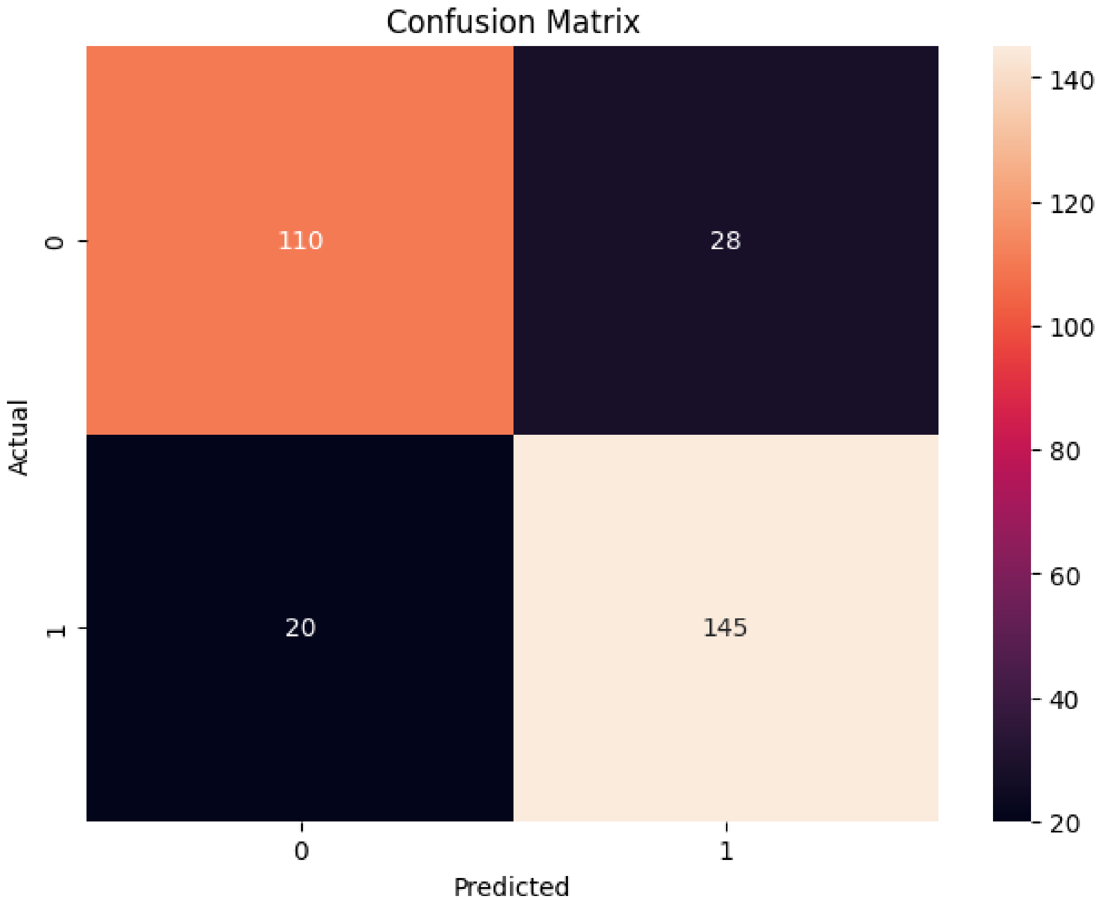
```
Accuracy: 0.8613861386138614
Precision: 0.847457627118644
Recall: 0.9090909090909091
F1 score: 0.862104631515783
```



[8]:
```
#PCA Extraction
X_train, X_test, y_train, y_test = reset(test, train, test.target, train.target)
X_train, X_test = Standardize(X_train, X_test)

# Best PCA feature extraction for this data
principal = PCA(n_components=10)
principal.fit(X_train)
X_train_pca = principal.transform(X_train)
X_test_pca = principal.transform(X_test)

# Training logistic regression model
model = LogisticRegression(random_state=0)
model.fit(X_train_pca, y_train)

# Predictions
y_pred = model.predict(X_test_pca)

disMetrics(y_test, y_pred)
```

```
conMatrix(y_test, y_pred)
```

```
Accuracy: 0.8415841584158416
Precision: 0.838150289017341
Recall: 0.8787878787878788
F1 score: 0.8420738302744872
```



Naive Bayesian Classification

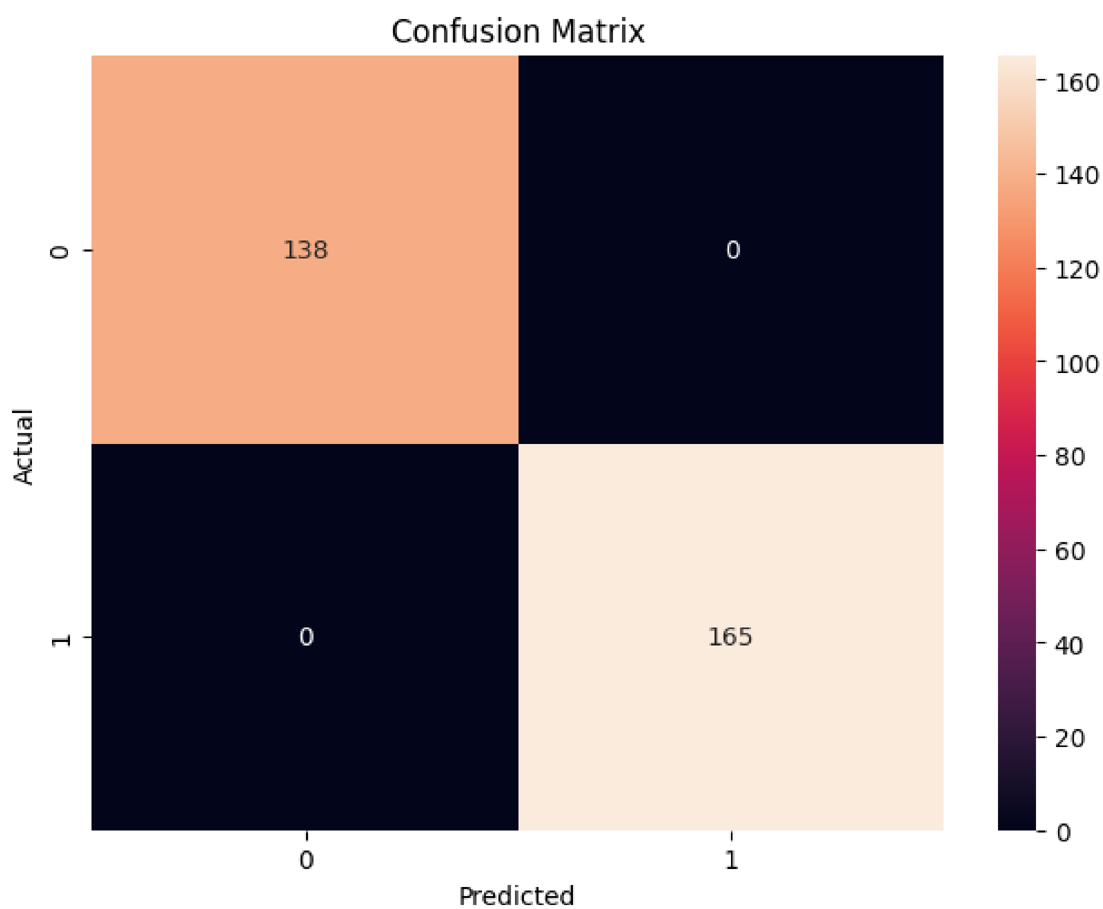
[9]:
```
#Naive Bayesian Classification
X_train, X_test, y_train, y_test = reset(test, train, test.target, train.target)
X_test,X_train = Standardize(X_test,X_train)
model = GaussianNB()
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

disMetrics(y_test, y_pred)
```

```
Accuracy: 0.834983498349835
Precision: 0.8362573099415205
Recall: 0.8666666666666667
F1 score: 0.8353436534129604
```

Decision Tree Classification

[10]:
```
#Decision Tree Classification
X_train, X_test, y_train, y_test = reset(test, train, test.target, train.target)
X_test,X_train = Standardize(X_test,X_train)
model = DecisionTreeClassifier(random_state=0)
model.fit(X_train, y_train)
y_pred = model.predict(X_test)

disMetrics(y_test, y_pred)
conMatrix(y_test, y_pred)
```

```
Accuracy: 1.0
Precision: 1.0
Recall: 1.0
F1 score: 1.0
```



References

Floyd, T. (2024). UNCC Repository [Computer software]. https://github.com/VoluSign/UNCC

Deeba, F. (2024). Introduction-to-ML Notebooks. https://github.com/Farah-Deeba-UNCC/Introduction-to-ML/

Dev Batra. (March, 2024). Heart Disease Prediction, Version 1. Retrieved from https://www.kaggle.com/datasets/devbatrax/heart-disease-prediction?resource=download.