Machine Learning Techniques for Heart Disease Classification

By: Sarah Petro and Jane Schneider

Website: https://sites.psu.edu/ds320project/

Table of Contents

Introduction

Our final project for DS 320: Data Integration involves using machine learning techniques for heart disease classification purposes, involving the integration of datasets with heart disease data from various cities. Using machine learning models to classify symptoms will help to diagnose a complicated and confusing disease with increasing accuracy, using mathematical relationships that go unnoticed by heart specialists.

Heart disease is one of the leading causes of death in the United States, per the CDC. Our motivation for the project is that the use of machine learning models to predict heart disease in patients has been studied as a possible solution to an unpredictable disease. Not only can machine learning models help predict the diagnosis of heart disease, it is also a solution to reduce and understand the symptoms related to heart disease. Using patient datasets can help narrow down crucial features, such as sex or heart rate to help researchers, and eventually doctors, improve performance of classification models to correctly and accurately diagnose heart disease patients. With faster and more accurate diagnosis, patients can receive treatment for the disease much faster which helps improve odds of solving symptoms and survival.

Our project is aimed to focus on the classification of heart disease based on three different datasets. The heart disease datasets are from leading data collectors and researchers on the topic named UCI Cleveland. We integrated three datasets from Cleveland, Hungarian, and Switzerland, to produce a more accurate model of Heart Disease Classification based on relevant factors such as age, sex, and symptoms.

Some challenges with this work is that classification of such a complicated disease is hard, which impacts the performance of the machine learning models. Heart disease can be caused by a lot of factors, such as genetics or environmental issues.

With such a difficult disease to predict, the model's performance can be impacted by small issues in the dataset, such as a couple irrelevant features. Researchers have been searching for ways to improve classification models in order to improve accuracy for diagnosis purposes.

Problem Statement

Our overarching goal with this project is to test different methods and models to find an accurate machine learning technique. These methods will include feature selection, integrating different data sources, and model testing with parameter tuning. After these methods, we will use a metric scoring summary to compare the performance of the models before selecting the most high-performing. Although accuracy is very important, other metrics must also be taken into account when selecting the most useful model. The goal of this project is to accurately classify heart disease diagnosis and symptoms since heart disease is one of the leading causes of death in the United States and is one of the most complicated diseases to diagnose.

Many researchers have embarked on the challenge to test and improve different machine learning models in the hope of accurately diagnosing heart disease, in benefit of the medical community. One prominent piece of work is "Classification models for heart disease prediction using feature selection and PCA" completed by Anna Karen Gárate-Escamila et al. in 2020. This research highlights the same dataset we will be using for our project and boasts between 98% and 99% accuracy for their machine learning model. The researchers used a method called Principal Component Analysis (PCA) to choose the most relevant features to train the model, which improved the performance of the model greatly. The research used a combination of six models: decision tree, gradient-boosted tree, logistic regression, multilayer perceptron, Naïve Bayes, and random forests. Other researchers have published a work called "Heart Disease Classification Using Neural Network and Feature Selection" by A. Khemphila and V. Boonjing in 2011 where they used feature selection and neural networks to predict heart disease classification. Their accuracy was lower at 82%, however this is another example of the well-researched and important topic of heart disease classification.

Data Preprocessing and Integration

Before testing different models, we first integrated data from three different locations, Switzerland, Cleveland, and Hungary, in order to create a larger dataset to train our classification models, a dataset entitled "heart_disease_uci". We integrated these three data sets on the similarity of their attributes, such as integrating attributes "fasting_blood_sugar" and "fbs" for example, which have different formatting, but overall the same purpose. Before continuing, we handled any missing values by dropping them, leaving us with 299 rows and 16 columns of data. Next, we preprocessed the

dataset by performing one hot encoding on categorical variables and changing numerical values into binary so they may be easily used for classification models. This includes processes like changing the sex attribute to represent 1 for men and 0 for women instead of keeping it categorical, which would pose no help to training our classification model. Similarly, if a patient had a fasting blood sugar value of true, it became a 1 or 0 if the value was false.

```
#perform one hot encoding on categorical variables and then change numerical 0,1,2,3,4 values into 0/1 binary for classification
dataset['thal'].replace({'fixed defect':'fixed_defect' , 'reversable defect': 'reversable_defect' }, inplace =True)
dataset['cp'].replace({'typical angina':'typical_angina', 'atypical angina': 'atypical_angina' }, inplace =True)


data_tmp = dataset[['age','sex','cp', 'trestbps', 'chol', 'fbs',  'thalch', 'exang', 'oldpeak', 'slope', 'ca', 'thal']].copy()
data_tmp['target'] = ((dataset['num'] > 0)*1).copy()
data_tmp['sex'] = (dataset['sex'] == 'Male')*1
data_tmp['fbs'] = (dataset['fbs'])*1
data_tmp['exang'] = (dataset['exang'])*1

data_tmp.columns = ['age', 'sex', 'chest_pain_type', 'resting_blood_pressure',
               'cholesterol', 'fasting_blood_sugar',
               'max_heart_rate_achieved', 'exercise_induced_angina',
               'st_depression', 'st_slope_type', 'num_major_vessels',
               'thalassemia_type', 'target']
data_tmp.head(15)
```

	age	sex	chest_pain_type	resting_blood_pressure	cholesterol	fasting_blood_sugar	max_heart_rate_achieved	exercise_induced_angina	st_d
0	63	1	typical_angina	145.0	233.0	1	150.0	0	
1	67	1	asymptomatic	160.0	286.0	0	108.0	1	
2	67	1	asymptomatic	120.0	229.0	0	129.0	1	
3	37	1	non-anginal	130.0	250.0	0	187.0	0	
4	41	0	atypical_angina	130.0	204.0	0	172.0	0	

Test-Train Split and Data Preparation

Before we went about training our models, we implemented the test_train_split package from sklearn to split our original data into training and testing data in order to test the accuracy, F1-score, precision, and recall of our models and ultimately choose the best model to classify heart disease given a set of attributes. Implementing this technique is vital to estimate the performance pattern of our models using sampling, that way we are able to make an informed decision about which model to select for final, full dataset training and prediction of target values. After splitting the original dataset, we were left with X_train (239 x 21), y_train (239 x 1), X_test (60 x 21), and y_test (60 x 1). Prior to training, predicting, and evaluating different models however, we normalized the data by using min-max normalization. This assigns every value to become a decimal value between 0 and 1 to use a common scale without distorting the differences in the range of values. This technique is often implemented by data scientists as a preparation for model training.

```
from sklearn.model_selection import train_test_split
y = data['target']
X = data.drop('target', axis = 1)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=0)
print(f'Shape of X_train: {X_train.shape}')
print(f'Shape of y_train: {y_train.shape}')
print(f'Shape of X_test: {X_test.shape}')
print(f'Shape of y_test: {y_test.shape}')
```

```
Shape of X_train: (239, 21)
Shape of y_train: (239,)
Shape of X_test: (60, 21)
Shape of y_test: (60,)
```

```
X_train=(X_train-np.min(X_train))/(np.max(X_train)-np.min(X_train)).values
X_test=(X_test-np.min(X_test))/(np.max(X_test)-np.min(X_test)).values
```

Models

The first model that we trained was logistic regression. Logistic regression was an obvious model to test because it uses regression analysis which finds the relationship between a dependent variable and one or more independent variables. We imported the LogisticRegression package from sklearn, fit the model with our split training data, and used the model to predict the Y test/classification given our X testing data. After finding our predicted y, we use metrics in order to measure the accuracy, precision, recall, and f1-score of our Logistic Regression model. The accuracy score that we received from our Logistic Regression model was 0.83.

Another model we chose to train and evaluate was the Decision Tree Classifier model. Decision trees are also high performing classification models because it breaks down a dataset into smaller and smaller increments while subsequently building a decision tree that determines the best predictors. Decision trees are common classification models, especially in cases such as this with a large amount of important features to consider. First, we created our model, then trained it with the X and Y training data, and set criterion to ‘entropy’ and the max depth to 5. Finally, we predicted our Y test from our X test data. After fitting and predicting, our model produced a metric summary which included accuracy, precision recall, and f1-score. The accuracy score was 0.78, without class weighting and 0.68 with class weighting.

We also used a Gradient Boosting Classifier, trained it with our training data and set its learning rate to 0.02, max depth to 3 and n estimators to 150. We decided to test the performance of a Gradient Boosting Classifier because this model uses ensemble learning which iteratively learns from each of the weak learners to build a strong model. Gradient boosting is a highly accurate model, especially for classification problems such as this one. First, we created our model, then trained it with the X and Y training data, and predicted our Y test from our X test data. After fitting and predicting, our model produced a metric summary which included accuracy, precision recall, and f1-score. The accuracy score was 0.83, which is similar performance to our Logistic Regression model.

Finally, the last model we explored was the Random Forest classifier, which fits a number of decision tree classifiers on samples of the training data and uses averaging to improve the optimal accuracy of the model. We used n-estimators of 150 and made

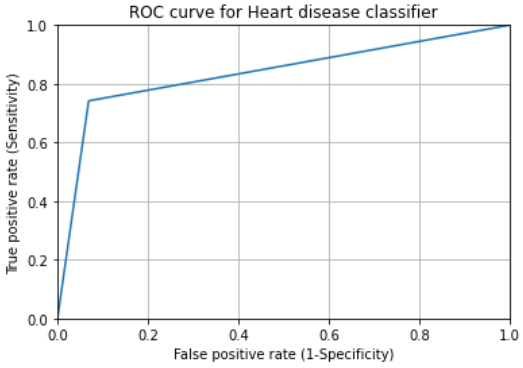
the class weight balanced, trained our model using our training data, then used it to predict the classification of our X testing data.

Results

Logistic Regression Results and ROC curve:

```
from sklearn.metrics import accuracy_score
print('The Accuracy Score is: ', accuracy_score(y_test,y_pred))
from sklearn.metrics import classification_report
print(classification_report(y_test,y_pred))
```

```
The Accuracy Score is:  0.8333333333333334
              precision    recall  f1-score   support

           0       0.77      0.93      0.84        29
           1       0.92      0.74      0.82        31

    accuracy                           0.83        60
   macro avg       0.85      0.84      0.83        60
weighted avg       0.85      0.83      0.83        60
```



Our logistic regression model had one of the high accuracy scores of our models we tested with an overall accuracy of 83.3%. The graph shown is the ROC curve for this classifier, which specifies the trade-off between the false positive rate and the true positive rate using different probability thresholds for binary classification.

Decision Tree (left) vs Decision Tree with balanced class weighting (right):

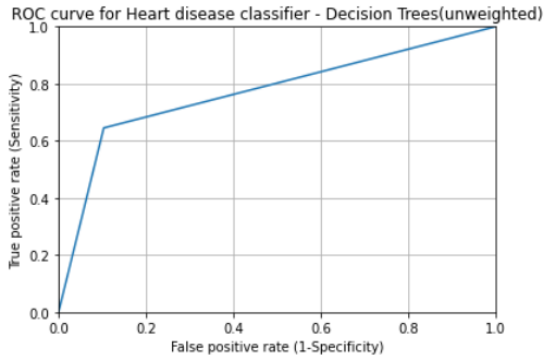
```
weights = {0:1, 1:0.5, 2:0.5, 3:0.5, 4:0.5}

clf = DecisionTreeClassifier(criterion='entropy', max_depth=5)
clf = clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print('The Accuracy Score is: ', accuracy_score(y_test,y_pred))
print(classification_report(y_test, y_pred))
```

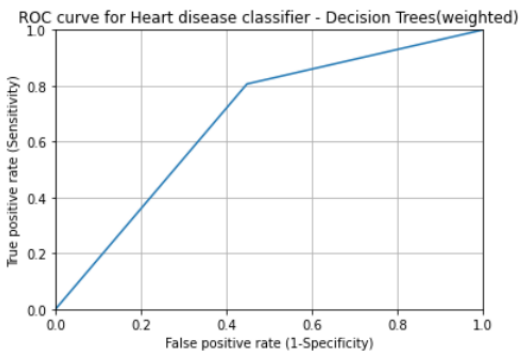
```
The Accuracy Score is:  0.7666666666666667
              precision    recall  f1-score   support

           0       0.70      0.90      0.79        29
           1       0.87      0.65      0.74        31

    accuracy                           0.77        60
   macro avg       0.79      0.77      0.76        60
weighted avg       0.79      0.77      0.76        60
```

```
[14] # Perform Decision Tree model with class weighting
     weights = {0:1, 1:0.5, 2:0.5, 3:0.5, 4:0.5}

     clf = DecisionTreeClassifier(criterion='entropy', max_depth=5, class_weight='balanced')
     clf = clf.fit(X_train, y_train)
     y_pred = clf.predict(X_test)

     print('The Accuracy Score is: ', accuracy_score(y_test,y_pred))
     print(classification_report(y_test, y_pred))
```

```
The Accuracy Score is:  0.6833333333333333
              precision    recall  f1-score   support

           0       0.73      0.55      0.63        29
           1       0.66      0.81      0.72        31

    accuracy                           0.68        60
   macro avg       0.69      0.68      0.68        60
weighted avg       0.69      0.68      0.68        60
```

ROC Curve:





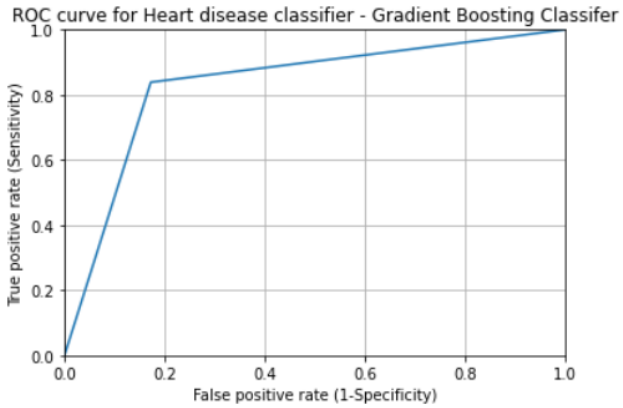
After testing the Decision Tree model with unbalanced and balanced classes, we found that our model achieved a higher accuracy score when classes were left unbalanced. According to our output statistics, our unbalanced Decision Tree had an accuracy score of 78% while the balanced Decision Tree had an accuracy of 68.33%. However, these scores are relatively low in comparison to the Logistic Regression, Random Forest, and Gradient Booster classifiers.

Gradient Boosting Classifier:

```
gradient_booster = GradientBoostingClassifier(learning_rate=0.02, max_depth=3, n_estimators=150)
gradient_booster.fit(X_train, y_train)
y_pred = gradient_booster.predict(X_test)

print('The Accuracy Score is: ', accuracy_score(y_test,y_pred))
print(classification_report(y_test, y_pred))
```

```
The Accuracy Score is:  0.8333333333333334
              precision    recall  f1-score   support

           0       0.83      0.83      0.83        29
           1       0.84      0.84      0.84        31

    accuracy                           0.83        60
   macro avg       0.83      0.83      0.83        60
weighted avg       0.83      0.83      0.83        60
```

ROC Curve:



The Gradient Boosting classifier performed well with an accuracy score of 83.33%. This is similar to the score of the Logistic Regression and Random Forest Classifiers. As a result of using ensemble learning methods, this model had the best precision, recall, and F1-score overall.

Random Forest Classifier:

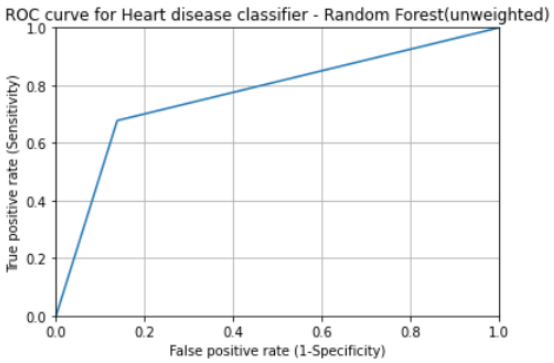
```
clf = RandomForestClassifier(n_estimators=150)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print('The Accuracy Score is: ', accuracy_score(y_test,y_pred))
print(classification_report(y_test, y_pred))
```

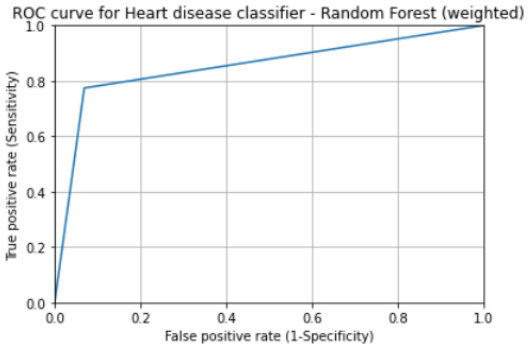
```
The Accuracy Score is:  0.8333333333333334
              precision    recall  f1-score   support

           0       0.79      0.90      0.84        29
           1       0.89      0.77      0.83        31

    accuracy                           0.83        60
   macro avg       0.84      0.84      0.83        60
weighted avg       0.84      0.83      0.83        60
```

```
clf = RandomForestClassifier(n_estimators=150, class_weight='balanced_subsample')
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

print('The Accuracy Score is: ', accuracy_score(y_test,y_pred))
print(classification_report(y_test, y_pred))
```

```
The Accuracy Score is:  0.8
              precision    recall  f1-score   support

           0       0.76      0.86      0.81        29
           1       0.85      0.74      0.79        31

    accuracy                           0.80        60
   macro avg       0.80      0.80      0.80        60
weighted avg       0.81      0.80      0.80        60
```

ROC Curve:





The Random Forest classifier with a balanced subsample performed a little worse than the classifier without balanced subsamples, with accuracy scores of 80% and 83.33% respectively. The random forest model without balanced subsamples also had a relatively high precision with a score of 0.79 and 0.89.

Results Statement

After reviewing the evaluation metrics for each of our models, we decided that the Gradient Boosting Classifier was the most effective at predicting the target output amongst the attributes selected from the heart disease UCI data. The Gradient Boosting Classifier shared a similar high accuracy score as the logistic regression and random forest model, yet outperformed them all on overall precision, recall, and F1-score. Specifically, we paid extra attention to the overall highest F1-score, as classification models are commonly evaluated by their F1 score as opposed to accuracy when classes are imbalanced.

We also set specific thresholds for classification, which is shown in each respective model's ROC curve. While thresholds can change, our conclusion would change on varying thresholds, therefore if this project were to be replicated with different thresholds, the results may show a different model with a higher performance than the Gradient Boosting Classifier.

Conclusion

In the completion of this project, our group saw how data integration is necessary for building and training large datasets so that our result could be better generalized to a larger population of differing genetic and environmental origin. Heart disease classification is inherently difficult to study, as a multitude of these genetic and environmental factors alter the risk of a patient developing heart disease. While research is ongoing, using the application of data science such as data preprocessing, data integration, classification model building and evaluation, and drawing conclusions from model statistics, medical specialists collaborate with data scientists to classify which patients are at risk using data collected from past patients. Some aspects that helped our group select the best classification model were performing one hot encoding of categorical variables, splitting our data into training and testing data, creating various models with tuning of hyperparameters, evaluating the models from metrics such as the F1-score, and summarizing our findings. While we have only grazed the surface of machine learning's application in heart disease classification, other works such as those done by researchers Anna Karen Garate-Escamila et al. have achieved model accuracy as high as between 98% and 99%, offering a model that could recognize patterns that otherwise go unnoticed by human capabilities.

Lessons Learned

To improve our project, some issues we could resolve include not removing the NA values in our dataset. Since medical data often has NA values, we ended up losing a lot of data for our train/test split. In improving our project, we can leave those NA values in and use a technique, like data imputation, to replace that data with a sampling method so that all of the observations from the dataset could be used.

We can also rank our risk factors for the medical community. Since risk factors can be very important in the diagnosis process, it is important for medical professionals to be made aware of the importance of the features and how to assist patients with risk factors, if they can be helped. Some risk factors, like age, cannot be changed, however many can be treated, such as smoking or diabetes. Ranking our features for importance would be part of a greater data visualization section we can improve our project with.

However, with our project, we also learned about data integration when we integrated our datasets, and integrating datasets on similar attributes, such as 'fbs' and 'fasting blood sugar'. We also learned about data preprocessing with removing NA values, encoding of categorical attributes, and min-max normalization for training classification models. Finally, we learned about model building, tuning, and evaluation by performing test-train splits to measure model performance and F1-score for evaluation of models with unbalanced training data.