Background

Coronary heart disease occurs when there is a buildup of plaque in one of the coronary arteries which reduces blood flow. A dataset from the Cleveland Clinic Foundation was used from the UCI Machine Learning repository which holds a list of medical attributes and whether or not coronary disease was found for 303 patients.

Goal

The goal was to create a tool based on this dataset to determine whether or not a patient has coronary heart disease.

Description of Data and Pre-Processing

The data consisted of 303 patients and 14 parameters. This was a processed and normalized version of the original dataset of 75 parameters. However, further feature selection was done by removing the features that had a Pearson correlation of less than .2 which left us with age, sex, chest pain type, fasting blood sugar, resting electrocardiographic results, maximum heart rate achieved, exercised induced angina, ST depression induced by exercise relative to rest, the slope of the peak exercise ST segment, number of major vessels colored by fluoroscopy, and results of a Thallium stress test. There were also 6 records with missing data that were removed when building classifiers. The target values (whether or not the patient had heart disease) were simplified to 0 and 1 rather than predicting the type of heart disease. Normalizing the dataset was tested but it did not improve the accuracy of any of the models.

Methods and Comparisons

The methods used were Neural Networks, Decision Trees, Support Vector Machine, Random Forest and Naive Bayes.

Artificial Neural Networks are composed of an input layer, hidden layers, and an output layer. Back propogation is used to create a more precise model within each hidden layer. The model we used was taken and modified for this dataset from https://www.kaggle.com/heyrict/clinicaldata-explanation-and-standard-eda-95. This method handles non-linear boundaries accurately because of the hidden layers that create new features that are used to separate classes. This method notably outperformed the rest of the classifiers in this case.

Decision trees are linear models that split based on minimizing the Gini impurity. These can be prone to overfitting so in this case, we split on nodes only with greater than 30 samples. A

notable benefit of decision trees is that they are easy to visually understand and help with the interpretation of the data.

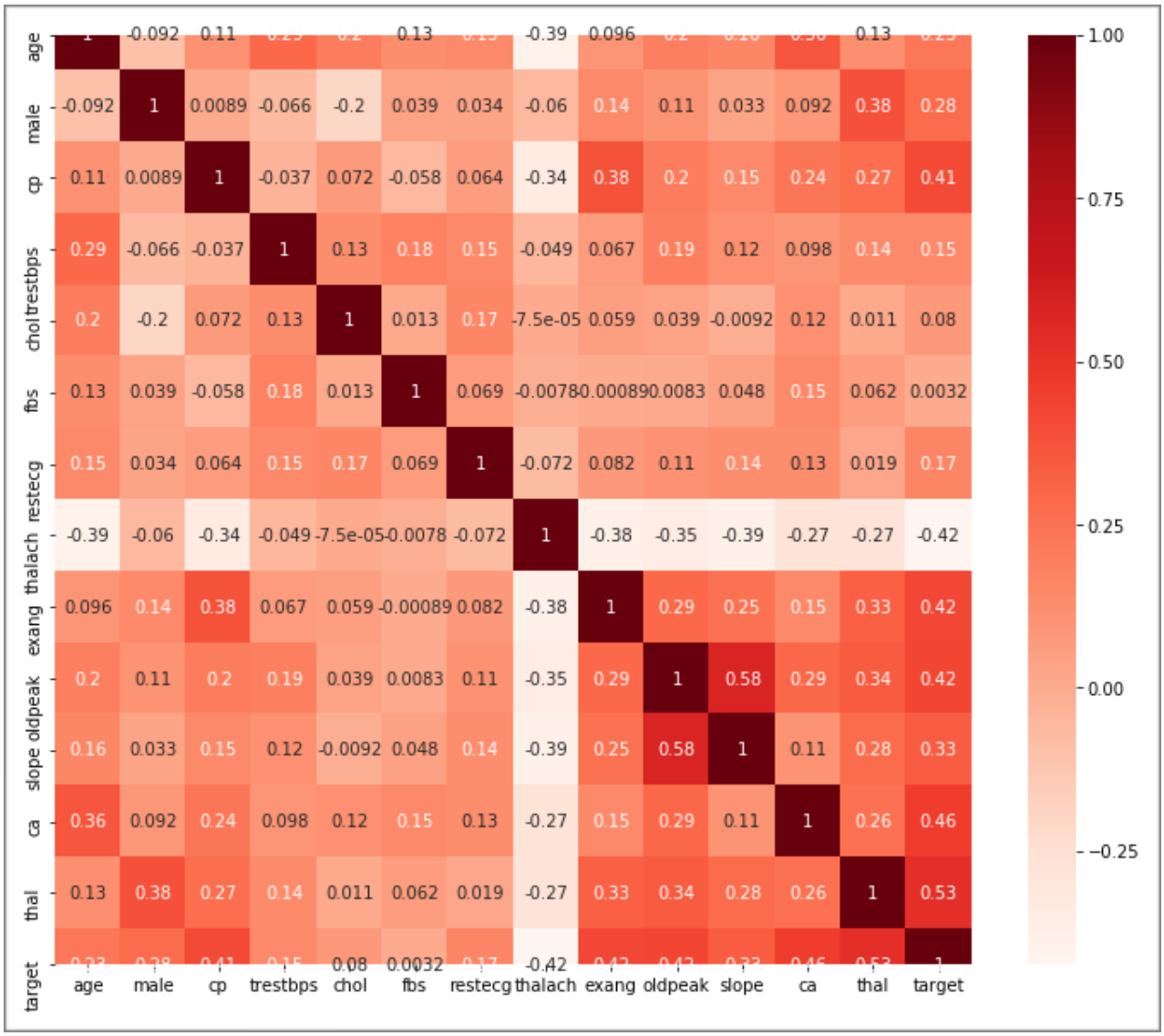
Random Forests combine many decision trees that are built from random samples of the training data and random selections of features are examined for each node that is split. Because of the random factor and that multiple decision trees are used to generated this classifier, it generally performs better than a single decision tree (as was the case in this project).

A support vector machine classifier is created by computing the most optimal hyperplane between the training data points. The best types of datasets to use these are generally small and with minimal noise. This is because the training time can be inefficient if there is a large amount of data. Similar to Neural Networks, these can also perform well with non linear boundaries.

Naive Bayes classifiers are generated by using Bayes theorem under the assumption that all of the categories in the data are independent from one another. This can be a problem as in our dataset and most dataset features rarely are completely independent.

Results

Feature Correlation Matrix



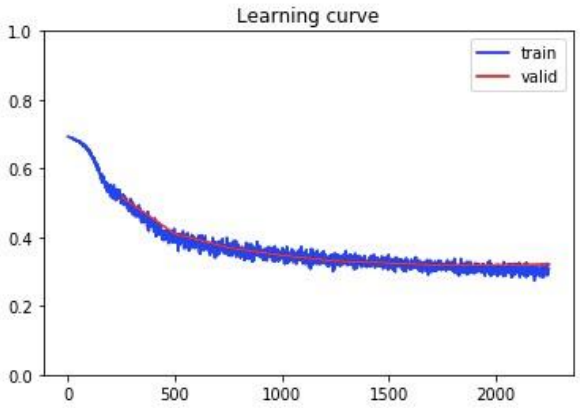
Neural Network Results

```
cm = pd.DataFrame(confusion_matrix(y_test, y_pred_lbl), columns=["T", "F"], index=
print("Accuracy = %.2f%%" % ((cm.iloc[1, 1] + cm.iloc[0, 0]) / cm.values.sum() * 1
cm
```

```
total loss in epoch 250 = 0.5104, validation loss = 0.5172, lr = 1.00e-04
total loss in epoch 500 = 0.4002, validation loss = 0.4095, lr = 1.00e-04
total loss in epoch 750 = 0.3509, validation loss = 0.3699, lr = 1.00e-04
total loss in epoch 1000 = 0.3389, validation loss = 0.3471, lr = 1.00e-04
total loss in epoch 1250 = 0.3243, validation loss = 0.3309, lr = 1.00e-04
total loss in epoch 1500 = 0.3051, validation loss = 0.3250, lr = 1.00e-04
total loss in epoch 1750 = 0.3139, validation loss = 0.3181, lr = 1.00e-04
total loss in epoch 2000 = 0.3132, validation loss = 0.3184, lr = 1.00e-04
total loss in epoch 2250 = 0.3100, validation loss = 0.3209, lr = 1.00e-04
Validation loss not improving for 500 epochs, stopping...
Accuracy = 90.00%
```

Out[2]:

	T	F
P	31	3
N	3	23



Support Vector Machine Results

In [3]:
```
# 2. Support Vector Machine


# update test and train data (it currently includes validation)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=21)

svclassifier = SVC(kernel='linear')
svclassifier.fit(X_train, y_train)
y_pred = svclassifier.predict(X_test)

print("Support Vector Machine Results")
print(classification_report(y_test,y_pred))

cm = pd.DataFrame(confusion_matrix(y_test, y_pred), columns=["T", "F"], index=["P", "N"])
print("Accuracy = %.2f%%" % ((cm.iloc[1, 1] + cm.iloc[0, 0]) / cm.values.sum() * 100))
cm
```

```
Support Vector Machine Results
              precision    recall  f1-score   support

           0       0.83      0.85      0.84        34
           1       0.80      0.77      0.78        26

    accuracy                           0.82        60
   macro avg       0.81      0.81      0.81        60
weighted avg       0.82      0.82      0.82        60

Accuracy = 81.67%
```

Out[3]:

	T	F
P	29	5
N	6	20

Random Forest Results

In [4]:
```
# 3. Random Forest

# Import the model we are using
from sklearn.ensemble import RandomForestClassifier
# Instantiate model with 1000 decision trees
rf = RandomForestClassifier(n_estimators = 1000, random_state = 42)
# Train the model on training data
rf.fit(X_train, y_train);
y_pred = rf.predict(X_test)

print("Random Forest Results")
cm = pd.DataFrame(confusion_matrix(y_test, y_pred), columns=["T", "F"], index=["P", "N"])
print("Accuracy = %.2f%%" % ((cm.iloc[1, 1] + cm.iloc[0, 0]) / cm.values.sum() * 100))
print(cm)
```

```
Random Forest Results
Accuracy = 80.00%
    T   F
P  27   7
N   5  21
```

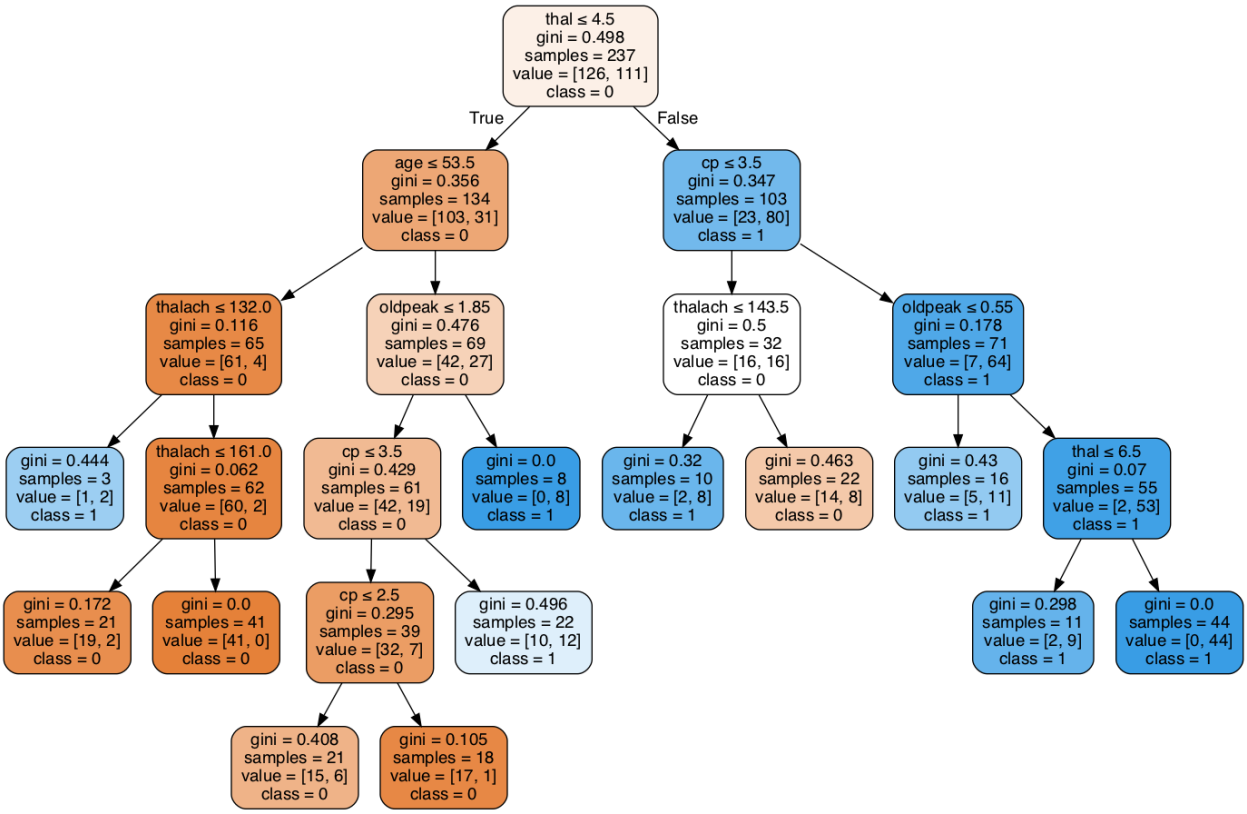
Decision Tree Results

Decision Tree Results

Accuracy = 73.33%

	T	F
P	25	9
N	7	19

Out[5]:



Naive Bayes Results

In [6]:
```
# 5 Naive Bayes
from sklearn.naive_bayes import GaussianNB

#Create a Gaussian Classifier
model = GaussianNB()

# Train the model using the training sets
model.fit(X_train,y_train)

#Predict Output
y_pred= model.predict(X_test)

print("Naive Bayes Results")
cm = pd.DataFrame(confusion_matrix(y_test, y_pred), columns=["T", "F"], index=["P", "N"])
print("Accuracy = %.2f%%" % ((cm.iloc[1, 1] + cm.iloc[0, 0]) / cm.values.sum() * 100))
print(cm)
```

```
Naive Bayes Results
Accuracy = 80.00%
    T   F
P  27   7
N   5  21
```

Method	Neural Network	Decision Tree	Support Vector Machine	Random Forest	Naive Bayes
Accuracy All Parameters	95%	73.33%	81.67%	83.33%	85%
Accuracy with less Parameters	93.33%	73.33%	81.67%	80%	80%

Discussion

This project is typical for a data scientist as it involved examining a large amount of data and using a variety of machine learning techniques and data processing and programming. It included understanding where the data came from and looking at models other people had created with it. Also, comparing different models is done frequently by data scientists as well as understanding how these models are generated rather than only knowing only how to use them.