Cai, Masek, McCarty

CS373 Final Report: Team # 26

Data

In 1988, researchers at the Cleveland Clinic Foundation published a dataset of 303 patients each with 76 attributes relating to demographics and cardiovascular health. Out of these 76 attributes, 14 were highlighted as primary indicators of heart disease based on previous studies. These include the patient's age, resting blood pressure, types of chest pain they might be experiencing, cholesterol levels, and ten other data points. Most importantly, it has a field representing the diagnosis of heart disease defined by a greater than 50% diameter narrowing in any major vessel in the heart. The goal of this project is to use machine learning algorithms to predict the diagnosis of heart disease in patients given the 14 specific attributes which are formally listed in the table below.

Feature title	**Index**	**Description (N - numerical, C - categorical)**
age	3	Age (in years) of patient (N)
sex	4	Gender of the patient (0-1) (C)
cp	9	Chest pain type (1-4) (C)
trestbps	10	Resting blood pressure (in mm Hg) (N)
chol	12	Serum cholesterol (in mg/dl) (N)
fbs	16	Greater than 120 mg/dl fasting blood sugar (C)
restecg	19	Resting electrocardiographic results (0-2) (C)
thalach	32	Maximum heart rate achieved (N)
exang	38	Exercise induced angina (C)
oldpeak	40	ST depression induced by exercise relative to rest (N)
slope	41	The slope of the peak exercise ST segment (1-3) (C)
ca	44	The number of major vessels colored by fluoroscopy (0-3) (C)
thal	51	Normal, fixed defect or reversible defect (3, 6, 7) (C)
num	58	Diagnosis of heart disease (0-1) (C)

Note: the features of this dataset are further described in dataset/heart-disease.names

Approach

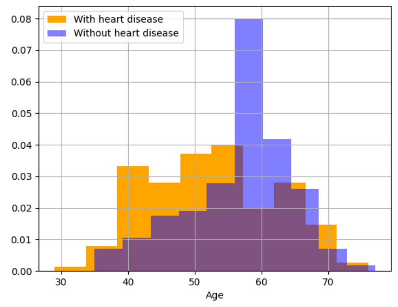
The two classification algorithms used for this project will be the k-nearest-neighbor (KNN) algorithm and tree learning classification. The KNN algorithm will require hyperparameter tuning of the number of nearest neighbors used. Whereas the tree learning algorithm will require tuning of the set max depth.

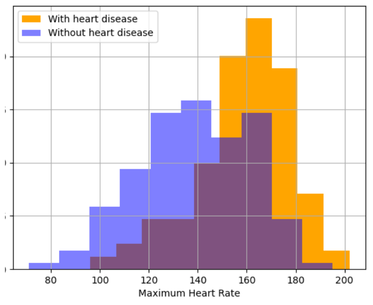
Initialization

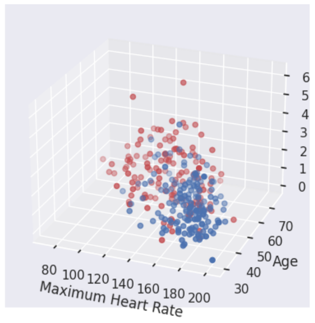
To start, a processFile() function was written which took in the original dataset "cleveland.data" and parsed it into a numpy array where each row represented a patient and each column represented one of the 14 primary features. This was compared to a preprocessed file, "processed.cleveland.data" to validate that the preprocessed file formatted the attributes as expected.

After this validation, the values in the preprocessed file were read into a data structure named "patients", where again, each row represented a patient and each column represented one of the 14 primary features.The order of the patients was then shuffled to remove possible, preset biases.

Next, a display() function was created to allow for easy visualization of the data. We ensured this was easily modifiable so that it can produce any wanted distribution. A few notable examples are shown here:







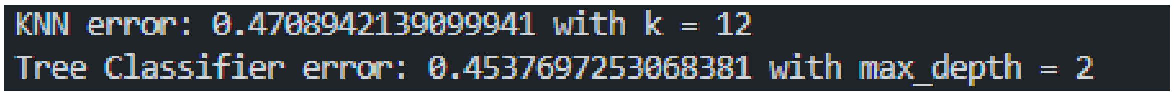
Training and testing

After thorough preprocessing and analysis, a trainAndTest() function was written that inputs the data (patients) and a model from the scikit learn library. This function trained the data on the model, then tested the model using 5-fold validation. This divided the patients into 5 groups. One of the groups was set aside and the model was trained on the other 4 groups, then the model was tested on the one set aside. This was repeated 5 times, so each group was tested on and the error returned by the function was the average error of these 5 repetitions.

Code was then included that called this function 15 times using scikit's k-nearest-neighbor implementation for each value of n_neighbors ranging from 1 to 20. Its error was recorded for each of these values and the n_neighbors value with the lowest, mean error was then printed. The same was done with tree classification with max_depth values from 1 to 14. As before, the max_depth value with the lowest error was then printed.

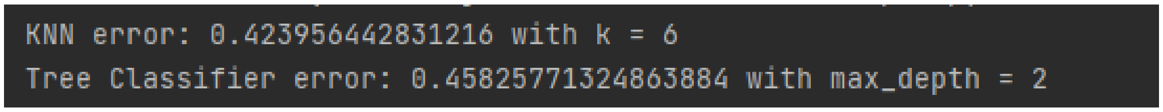
Initial Improvements

At this point, the results were less than ideal. The KNN algorithm had about a 47% error and the tree classifier had an error of about 45%. Both of these numbers were too close to the theoretical worst 50% error, so some more preprocessing was needed to be done.



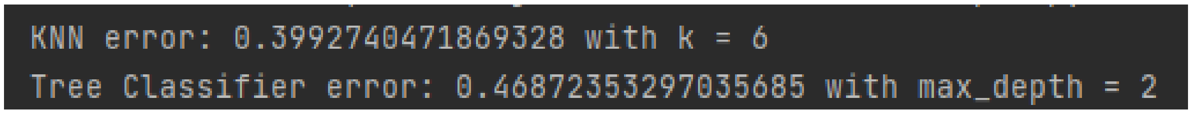
```
KNN error: 0.4708942139099941 with k = 12
Tree Classifier error: 0.4537697253068381 with max_depth = 2
```

The first thing done to improve performance was to scale continuous features to be in a range from 0 to 1. This was to standardize these features by removing the median and scaling the data according to the quantile range in statistics. This also helps with reducing the impact of outliers in the dataset. This was accomplished with scikit’s RobustScaler class and was applied to the data. This brought the KNN error down to about 42% but had little effect on the tree classifier. This is logical, because the tree classifier treats every feature separately so scaling would have little effect.



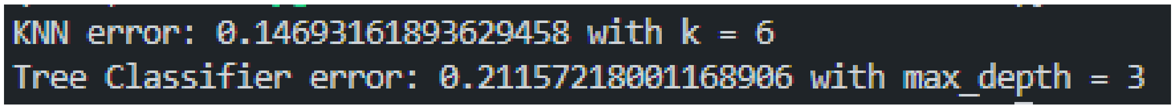
```
KNN error: 0.423956442831216 with k = 6
Tree Classifier error: 0.45825771324863884 with max_depth = 2
```

The next action to improve performance was to split categorical variables into separate boolean columns. This is to prevent the algorithm from assuming a natural relationship within the values of categorical features. This was done using scikit’s OneHotEncoder class to encode categorical features as a one-hot numeric array. This brought the knn error down to about 40% but also had little effect on the tree classifier. This is also logical because a tree classifier would divide the data based on the values of the categorical variables and tend not to put weight on the ordering of the variables.



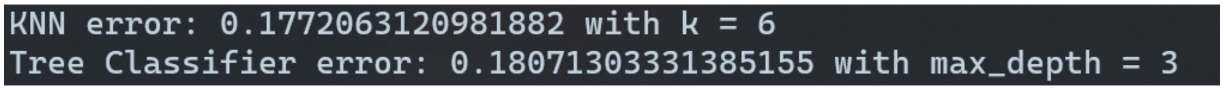
```
KNN error: 0.3992740471869328 with k = 6
Tree Classifier error: 0.46872353297035685 with max_depth = 2
```

At this point, it started to become obvious that an assumption we made about the data was incorrect. We combed through the data manually and then realized the output column was from a range of 0 to 5, not 0 to 1. After rereading the dataset description, we realized that every non-zero value was supposed to represent a positive result for heart disease. So we added an additional line to binarize the output column. Now our results are looking a lot better:



```
KNN error: 0.14693161893629458 with k = 6
Tree Classifier error: 0.21157218001168906 with max_depth = 3
```

After performing another code review, we realized that we were transforming the data prior to splitting it into testing and training sets. This would mean the mean would be found between the two sets, effectively mixing information between the two. After changing the code to transform the data after it is split, we received the following result:



```
KNN error: 0.1772063120981882 with k = 6
Tree Classifier error: 0.18071303331385155 with max_depth = 3
```

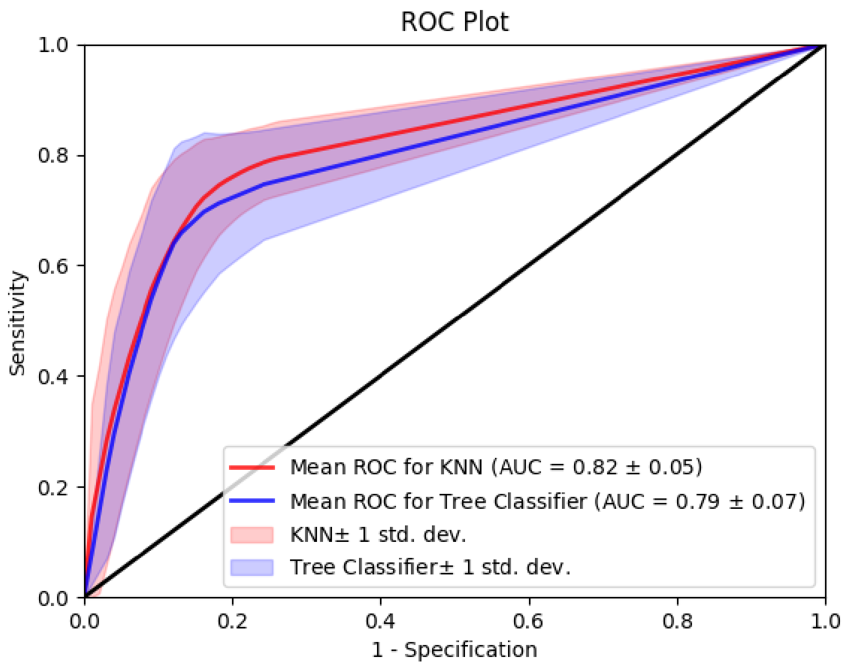
As expected, the error increased because we did less fitting to the testing data, however this creates a more effective model because overfitting is reduced.

Experimental Results

In an effort to analyze the performances of each classifier, we collected the following results.

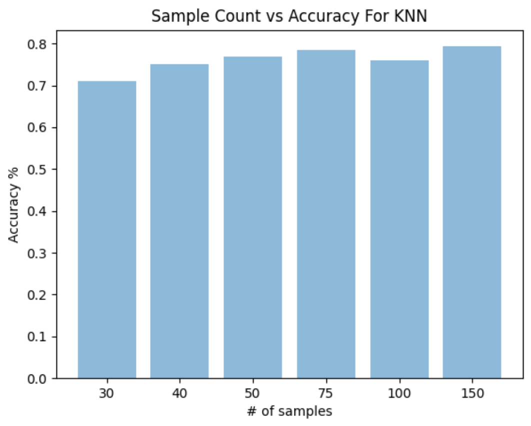
ROC Curves

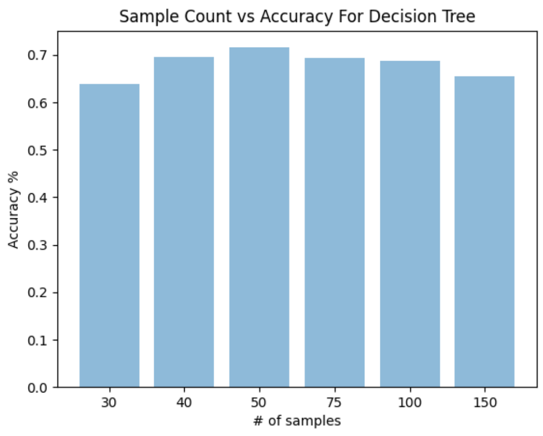
While initially tuning the hyperparameters for each model, the count of “True Positives”, “True Negatives”, “False Positives”, and “False Negatives” were recorded and used to calculate sensitivity and specificity for each fold and hyperparameter value. These calculations were then used to find the overall ROC plot for both classifiers:



Number of samples vs. accuracy

We obtained plots of the number of samples versus accuracy using different sized subsets of the dataset. This was accomplished by repeatedly choosing a sample size n, then finding the average, prediction accuracy associated with that sample size. Predictions were made using the hyperparameters with the lowest error found in tuning. Results show an accuracy of ~70-80% for KNN and ~60-70% for Decision Tree.





Discussion of Results and Conclusion

The experimental results indicate that both classifiers perform much better than a worst-case of random. The ROC plot seems to show that a KNN classifier may have slightly better performance and consistency than a decision tree, given it's higher AOC and decreased deviation. However the difference is marginal and likely to be roughly equivalent. Likewise, the number of samples vs. accuracy plot seems to reiterate the conclusions found in the ROC plot. KNN shows consistently higher, but marginal, improvement over the decision tree. If implemented in a practical scenario, KNN would likely be the choice between the two classifiers. Given that an accuracy of 70-80% was achieved, this could realistically be used in a number of ways. As healthcare technology improves and more 'at-risk' individuals are being equipped with sensors to monitor their health, such a classifier would be able to identify when a patient needs urgent care. Additionally further testing of these classifiers can be performed to better identify which features put individuals at higher risk for heart disease.