Sahil Desai

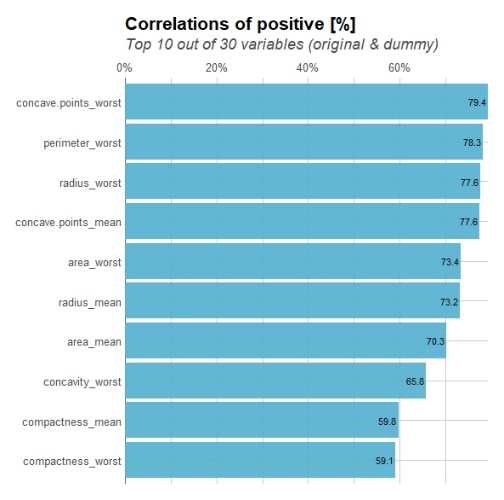
Ebhrahim Mortaz

MGT 388

5/9/2020

**Final Project Report**

This project was a great introduction to testing different machine learning algorithms. I did run into some problems we discussed in class such as overfitting (which I will discuss later in the report) which helped me gain deeper understanding on how to tune different algorithms. This dataset was mainly numerical, with one target variable being the classifier for cancerous and noncancerous. I will start by discussing my original plan with the EDA analysis.

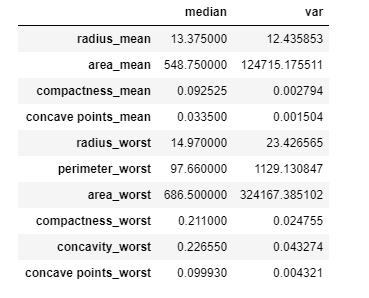
To begin, I found that there were a lot of numerical variables to explore. First, I dropped the ID column since that yields no value. Then, I wanted to find the variables that were most highly correlated with the diagnosis column. Since this was the beginning of the project, I went straight to R and created a plot that pits diagnosis with its most highly correlated variables (do not worry! This is the only time I used R in this project since it was the beginning). This was created using the Lares package.

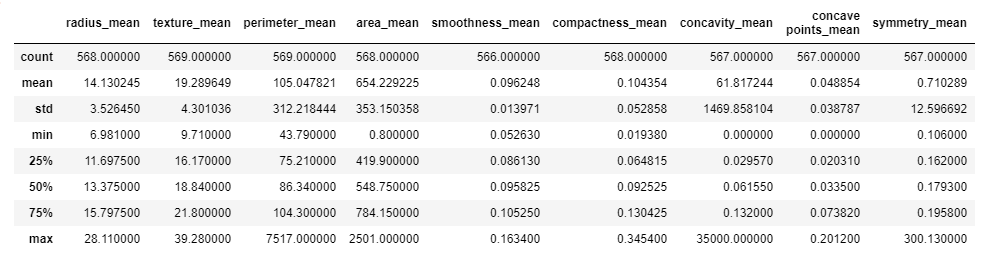
Alternatively, there is a feature selection technique from sklearn that plots a similar graph

A screenshot of a cell phone

Description automatically generated

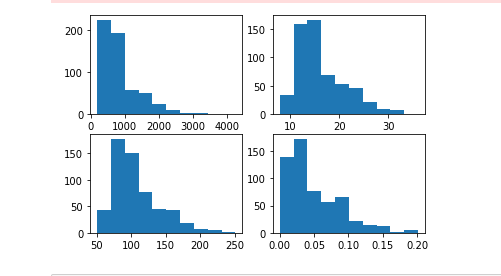
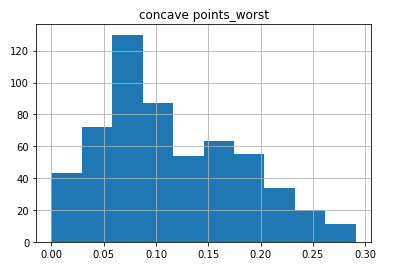
From here, it was simple EDA such as using the describe function to see how the data for each column was distributed. The count for benign (B) and malignant (M) tumors was also calculated with B having 357 counts and M having 212 counts. In addition, I also found median and variance of the top correlated columns and incorporated it into an easy to read data frame.



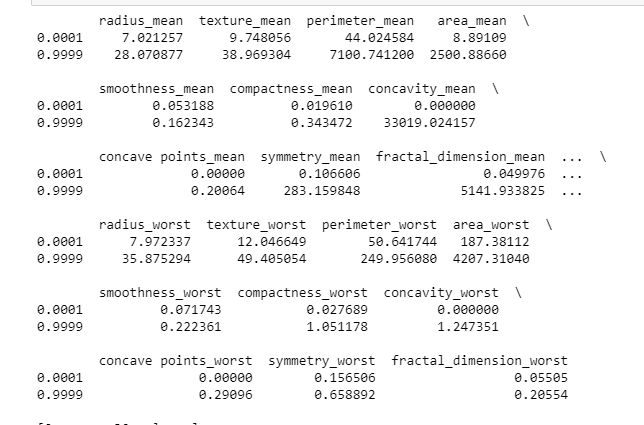


I did a quick checkup on the NA values in the dataset, and surprisingly there were not many listed among the features (under 30 NA values total). I decided it would okay to completely remove rows with NA’s for the machine learning portion as this would not result in much data loss.

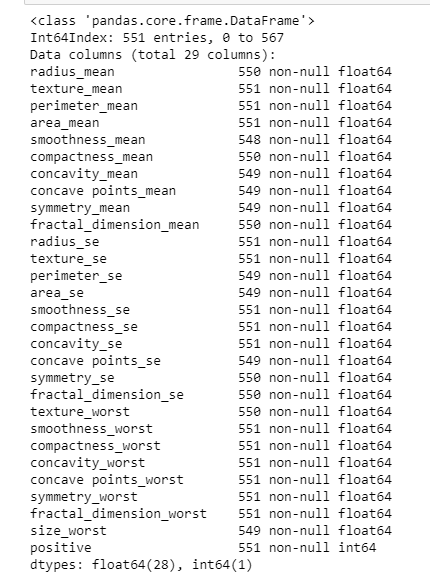
To check for data distribution, I decided to use simple histograms over the only the top 5 most correlated values. Below are four of those plots, with my Python program which one is which. The last plot is inherently the most correlated feature of the bunch

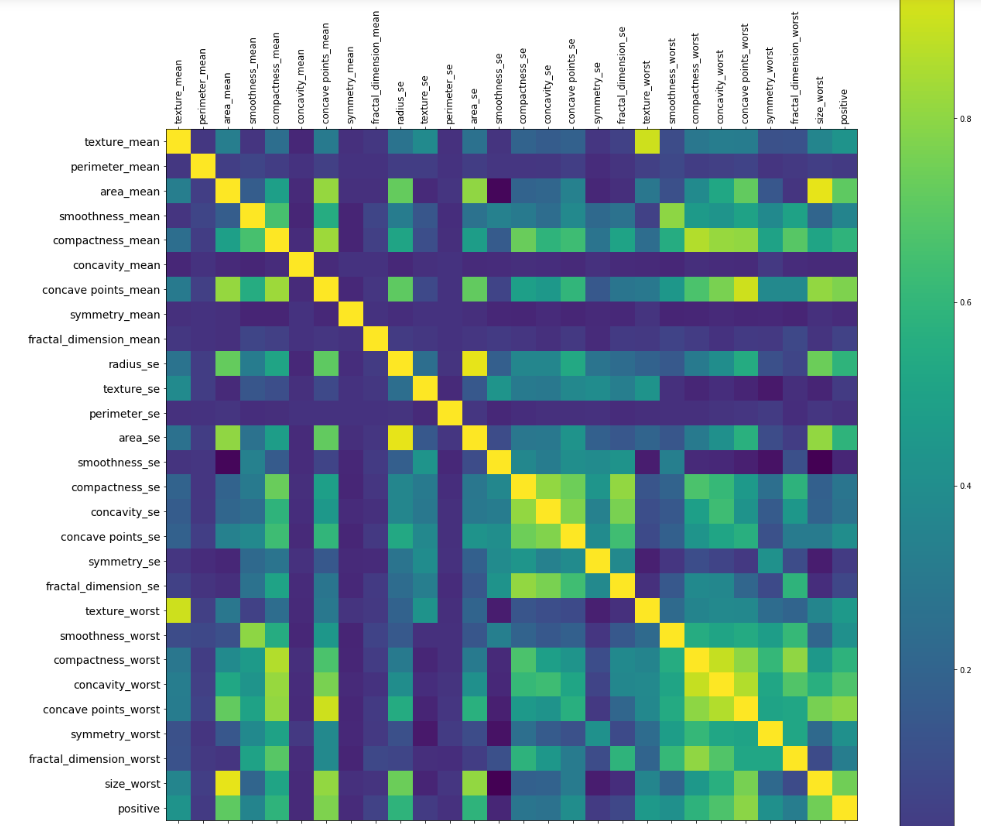


As for identifying outliers, I used a quantile function and initially wanted to filter out values that were above and below the 95% intervals. However, this resulted in too much data loss, and I found that the outliers in the dataset were more extreme. I filtered out the values above and below the 99.99% intervals among the feature columns. We can see from this output some very glaring outliers that would negatively impact the machine learning models



After I removed these, I decided to engineer a new variable called ‘size\_worst’ which is a culmination of adding area\_worst, radius\_worst, and perimeter\_worst columns. From here, I decided to drop these three as I felt it would have been redundant data that would not benefit my models at all. To finish off the EDA, I added a dummy variable for the diagnosis column so my machine learning models can have a target variable. B would be coded as 0 while M would be coded as 1 for each observation. I then dropped the diagnosis column as it was no longer needed and created a heat map correlations matrix to finish this part of the project. To begin the machine learning portion, I ran the info function to double-check I had the columns I wanted.

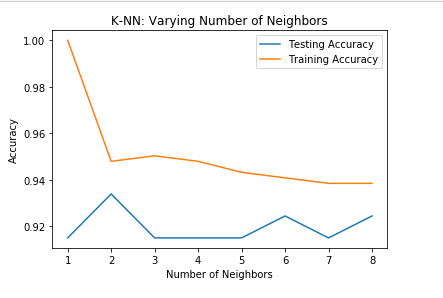




**Machine Learning**

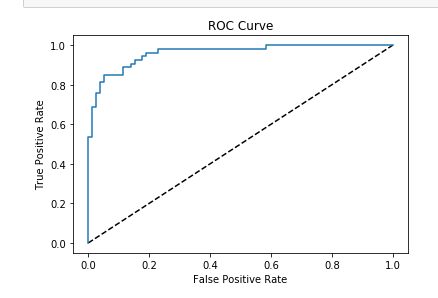
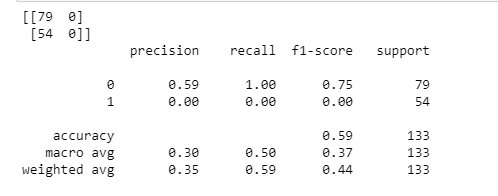
**\***Before we start the machine learning algorithms, I filtered out all NA’s values from the data set.

*KNN* was the first method to be implemented. I created a code where I was able to get each neighbor from one through eights’ accuracy score using KNN. My training data consisted of 80% of the data and testing was 20%. I also stratified so the target variable was proportional through the split. The most accurate ending up being where the neighbors parameter was set to two. Below is the graph displaying test and train accuracy.



One thing to note is that my testing accuracy was stagnant until increasing the number of neighbors past six. After that is started to decline.

*Logistic Regression* was the next method to be utilized. I used a test size of 25% and train size of 75%. This time, I decided to normalize the data since some of my features had different ranges than others. When I implemented the logistic regression model without normalization, I found overfitting to be a problem. That is probably because logistic regression is not going to make assumptions on the distributions of my data. After implementing this, I came with a lower accuracy score that I could tune. Below is the confusion matrix, classification report, and ROC curve.

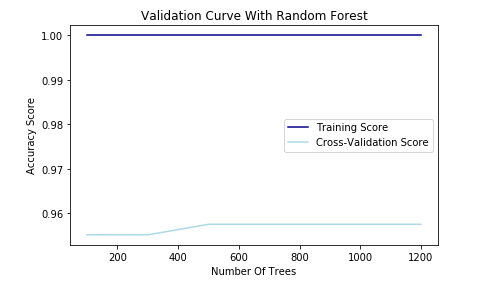
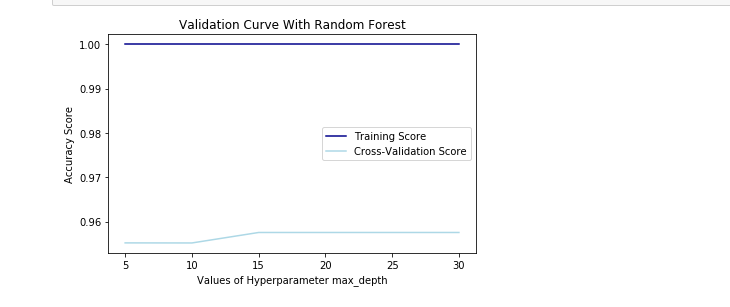


From here I used cross-fold validation (cv=5) which increased by accuracy immensely.

The next algorithm I incorporated was a *decision tree*. Undoing the normalization from the previous model was the first step. I went back to making my test size 20% and my train size 80%. I ended up coming up with 92% accuracy on the test score on the initial run. From here, I instilled different accuracy scoring methods (gini and entropy). To improve my accuracy, I did a randomizedsearchCV (cv =5) with suggestions to parameters such as max depth, criterion, max features, and min samples leaf. This contributed to a slight raise in accuracy score from 92% to 93%.

My next algorithm was the implementing *random forest*. My training data was 20% and my testing data was 80%. I came away with an accuracy score of 93%. Using GridSearchCV for tuning ended up being computationally expensive, but I was able to find the best hyper parameters. I also plotted validation curves to get a better look on how the CV score changes with different parameters.

I was able to increase my accuracy. Below are the validation curves.



The final technique I used was gradient boosting. My training data was 80% and testing data was 20%. I came away with 94% accuracy off the default. Using GridSearchCV (cv = 3), I instilled two hyperparameters (learning rate, n estimators) that I wanted to see the best values for. After finding them out from the grid search, I tuned them into my model which raised my accuracy to 95%.