CPSC 486

Review of Internship on Machine Learning Models

The idea for this internship was to take a dataset from a reputable site and try and run different models on that data for the best results. The site I found most helpful for both learning models and following examples was www.kaggle.com. From this website I was able to choose the dataset “200 + Financial Indicators of US Stocks (2014-2018)” which was created by Nicolas Carbone [1]. This was a great initial data set because there was an abundant number of features to use and it had a flexible set of paths I could take to run models on it. Since this was my first attempt both at using python and machine learning, my goal was to try and train the models for classification into stocks that I would recommend buying or not buying at the beginning of the year.

The first step with this dataset is to try and figure out an objective that I would be able to accomplish in a timely manner, while still learning new skills to create these models. Since the total set consisted of 5 different years of data, at my skill level I decided it would be better to try and analyze just one year. I choose the 2017 dataset because it consisted of 4,960 rows of stocks and 224 columns of features which left plenty of data to work with. The options for what I could do with this data was broken down into using the “Class” feature as my target or using the “Price Var [%]” for my classification. I chose the “Class” feature because it was a 1 or 0 representing if I should or should not buy a stock. The Price Var was a numerical representation of this and would later be a learning point in my program. With a goal in mind I was able to move ahead to cleaning data and trying to run models on a relatively pure dataset.

To better understand the data types I was working with I used the command “dtypes” on my data frame to get a count of each type of data. For the most part the features were broken into floats except for the “Sector” and” Id” as objects and the “Class” column as an int. The Sector column was all I had to worry about since the others were not used in the actual features. I used a label encoder to transform the Sector data to the proper format which would allow me to run different visualizations on the data frame. The first type of visualization I would run would show me how the class distribution looks before anything is done to the dataset.

Figure 1.1

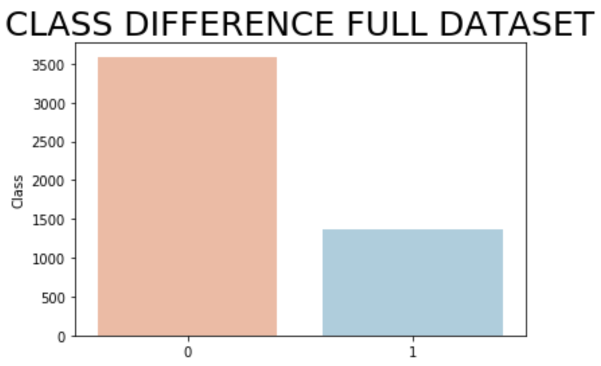


Figure 1.2

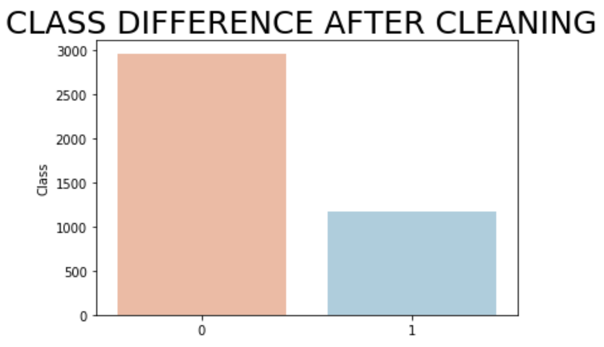


Figure 1.1 is a demonstration of what the classes look like before any work is done to the data. This figure shows a heavy dominance of the 0 class while having less than half for the 1 class. This is important to keep in mind later when we are using the train test split function to break up our data. To solve this issue, we stratified the target feature letting our train and test sets have a similar distribution. If you look at figure 1.2 you can see that after cleaning the data, we were able to get these classes a little bit closer, but still not close enough to not use stratification.

Although I was just focused on the 2017 data set, I did check the other datasets for this class distribution, and they seemed to have a similar difference in classes.

The next way that I choose to breakdown the features was using the "Sector" feature I had previously used an encoder on. It was important for me to breakdown this feature because stocks are not always about the numbers and this was a good learning tool to see what I was dealing with.

Figure 1.3

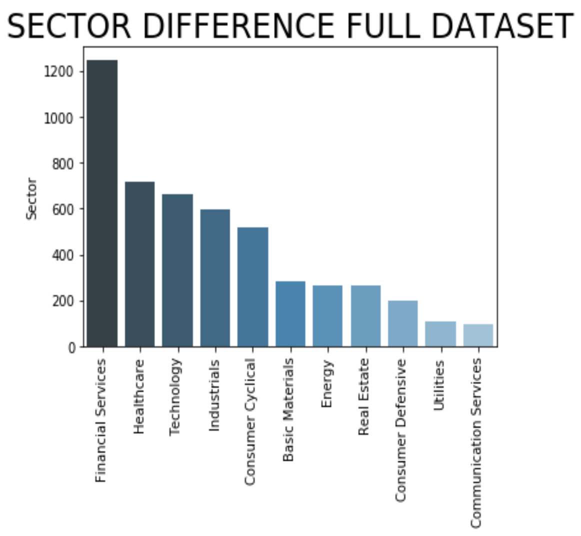
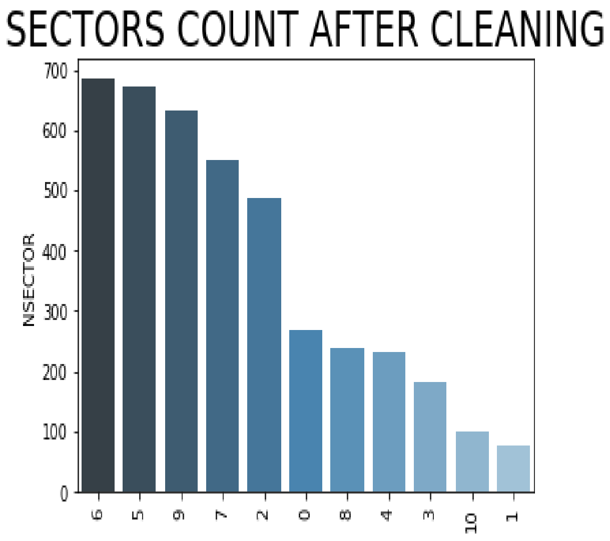


Figure 1.4



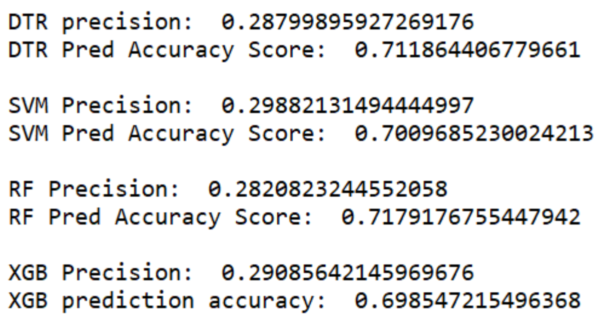
In figure 1.3 we can see a major difference happening with the sectors with the Financial Services holding more than half of the stocks we are using. This information would helpful when we run our models because depending how that sector is doing at the time, it may have a heavy influence on our results. As you can see in figure 1.4, I was able to reduce the number of stocks we choose to use which gave us a better distribution of our data over the sectors.

To get the resulting dataset in figures 1.2 and 1.4 I used a couple of techniques to clean the features. The very first thing I did was make sure the features and stocks, (columns and rows) did not hold too many NAN values. If there was more then a 50% NAN values in either of these that feature, or stock was dropped as it would not lead to valuable information. Another major tool I used was a was a correlation heat map to see which features tended to give me the same information as using both would be a waste of time and resources. With the amount of data I had this heat map is to large to fit here, but I ended up dropping features that had a correlation of .8 or more. After all this cleaning the initial models I would run would have a data set of 4130 rows with 219 features.

The models I decided to use for this dataset were the DecisionTreeClassifier, Support Vector Machine, RandomForstClassifier, and the XGBoost model. These models are commonly used and beginner friendly which was good for learning with a lot of information to research on them. To get these models running at their peak performance on this dataset, Nicolas Carbone offered a great section of his Kaggle notebook for tuned parameters. [2] I was able to use most of these parameter functions to find the optimal parameters for each type of model. These tests are good for finding the right parameters but are also take a substantial amount of time to complete based off the cross validation and model itself.

After the models were all running properly it was time to run some analysis on the models. There were a couple of tests I wanted each model to produce. I ran a model score and a precision score on every test to see how effectively the models were classifying.

Figure 1.5



```
DTR precision:  0.28799895927269176
DTR Pred Accuracy Score:  0.711864406779661

SVM Precision:  0.29882131494444997
SVM Pred Accuracy Score:  0.7009685230024213

RF Precision:  0.2820823244552058
RF Pred Accuracy Score:  0.7179176755447942

XGB Precision:  0.29085642145969676
XGB prediction accuracy:  0.698547215496368
```

Figure 1.5 is a representation of the output of the code I got from running these models. I had run these same models on the dataset before cleaning it and had gotten lower numbers. This meant that my models were improving and the RandomForrestClassifier seemed to be performing the best on this dataset for me.

Figure 1.6

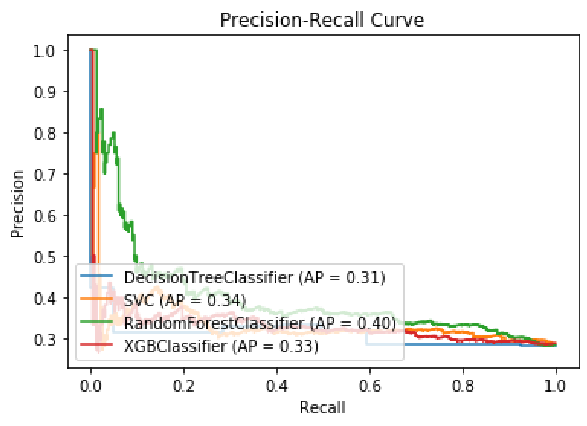
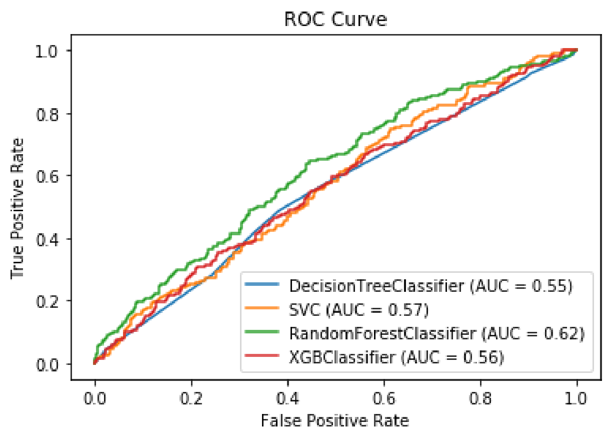


Figure 1.7



Both figures 1.6 and 1.7 are visualizations of how the models are performing based off different factors. The Precision-Recall Curve and the Receiver Operating Curve are both used to judge how well the classifier is working. Both graphs are giving us relatively low values indicating that our classifiers still need some work. Figure 1.6 is indicating that we are only reaching high recall at very low precision rate, while Figure 1.7 is showing the models at around 50% in predicting both true positives and false positives. Although we have seen an increase in the prediction accuracy, there Is still work to be done to the dataset to try and get more accurate results.

One solution to this was to reevaluate the features I was using for the dataset. There was still a ton of data to work with and maybe a better representation of this dataset would help. Python and more importantly sklearn provide great tools to be able to reduce the number of features on your dataset for optimal performance. The tool I decided to use was the recursive feature elimination better known as “RFE”. This tool allows you to set the number of features you would like to use, ranks them accordingly by importance and then creates the model to run on the data. While using this tool I had chosen the number of features to us to be 75 originally and then worked my way down to 50. This test takes a lot of time to run and I was not able to use it on the support vector machine model because of computer strength.

Figure 1.8

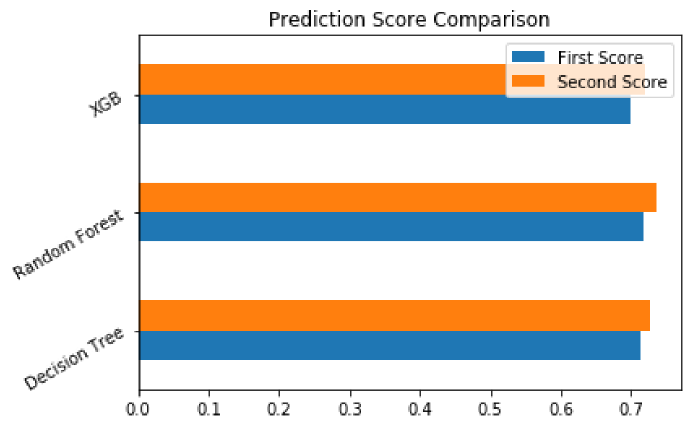


Figure 1.8 shows the prediction scores of the three models from before using RFE (blue bar) and after using RFE (orange bar). This graph indicates that the models are having higher prediction scores with the more specific number of features selected from the RFE tool. In the future it would be interesting to run this tool on a wider range of features.

Overall, this internship has taught me a lot of different lessons and has given me a core set of skills to work with in the future. I enjoy working with data and like to see the models improve over time. For the future I would like to construct different models and tests on this dataset or find some way to compare the different years all together. One invaluable lesson I learned when working with data is that you need to understand the data you have to get the results you want. I ran many models with the Price Var feature still in the dataset which led me to some off the wall answers. This internship has also showed me the importance of considering the tests you are running and the complexity of these test. I ran multiple tests for hours but did not have the computing power to complete them. It is important to understand what these tools are doing and the requirements to complete them. I have learned quite a bit about training different models and what it takes to increase the results of these models. In the future I would like to use these models to solve different problems and see how far I am able to push them.

Reference

[1] N. Carbone, "Explore and Clean Financial Indicators Dataset," February 2020. [Online]. Available: https://www.kaggle.com/cnic92/explore-and-clean-financial-indicators-dataset. [Accessed 20th March 2020].

[2] N. Carbone, "200+ Financial Indicators of US Stocks (2014-2018)," February 2020. [Online]. Available: https://www.kaggle.com/cnic92/200-financial-indicators-of-us-stocks-20142018/kernels. [Accessed 20 March 2020].

[3] N. Carbone, "Beat the Stock Market: The Lazy Strategy," 1 January 2020. [Online]. Available: https://www.kaggle.com/cnic92/beat-the-stock-market-the-lazy-strategy. [Accessed February 2020].