Machine Learning Engineer Nanodegree

Capstone Project -- Portfolio Optimization

Giordanni Piguing

September 20, 2017

I. Definition

Project Overview

Financial models have been used for years to better understand market behavior and make financial portfolios profitable. Years of historical stock prices are available for the public which are suitable for machine learning to generate such financial models.

The equity forecast [1] by Nikola Milosevic predict whether some company's value will be 10% higher over a period of one year using machine learning. The paper has motivated me to write my own model. Unfortunately, I don't have access to all the company information that he used as his features. I'll just use whatever information will be easily accessible to me.

Using the machine learning techniques and algorithms I have learned from Machine Learning Engineer Nanodegree, I will be generating a stock price predictor model. It will help predict when to buy or sell a certain stock base on its current performance. Using that model, I will generate a portfolio optimization that automates trading and see how it will perform over a specific time period.

Problem Statement

The stock market is very volatile and unpredictable. Although there are plenty of information available per stock, such as earnings, market cap, profit/expense ratio, news, etc, it is still hard to predict where the stock price will go. If it was easy, then everyone could have made millions from stocks already.

For this project, I will only be using the daily trading data over a certain specified date range as training sets. The date would go from 04/01/2014 to 5/31/2016, about 2 years of data. The data will contain features such as opening price (Open), highest day price

(High), lowest day price (Low), closing price (Close). The model will then predict whether to buy or sell certain stocks, on a daily basis, for a certain specified date range. The supervised training target will be generated by taking the 'Close' price and shifting it back. Basically the models will try to predict the next 'Close' price of the stock. If the model thinks the price will go up tomorrow, it will recommend buying the stock today. On the other hand, if the model thinks prices will go down tomorrow, it will recommend to sell the stock today. The models will determine a pattern, based on the existing features. Since I cannot have the data for the future, I'll be using old data. The testing date will go from 6/01/2016 to 7/31/2017. It will also follow different stocks than the training sets, to make sure the model didn't just over fit or follow the trends from the training sets.

Multiple supervised learning algorithms such as Decision Tree Classifier, Gaussian Naive Bayes, AdaBoost Classifier, and Artificial Neural Networks will be evaluated in order to determine the best one. These 4 models will have 120 different versions, based on different target ratios, buy, and sell prices. Thus a total of 480 different models will be generated. I will discuss further how these models are different later on. A benchmark model will also be used, to measure the performance of the supervised models. The benchmark model will just hold the stock for a year. I will then compare the portfolio of the models against the benchmark. This will be discussed thoroughly later.

Metrics

Using cross validation and train test split, I will be calculating the accuracy and F-score of each of the models. The accuracy will help measure how well the model is doing in predicting whether the price will go up or down the next day, based on the training sets. The F-score is more important than the accuracy though. If the data is imbalance, the accuracy can be misleading. As an example mentioned by Cisneros [5], if there was 1,000 electric guitar images and 100,000 acoustic guitar images. If from the 1,000 electric guitars only 493 were labeled as electric (True Positives=493), the rest were labeled as acoustic (False Negatives=507). On the other hand, from the 100,000 acoustic guitars 98,983 were labeled as acoustic (True Negatives=98,983) and only 1,017 of them were labeled as electric (False Positives=1,017). For these example the accuracy is 0.98 but the F-score is only 0.39. Thus, accuracy could be misleading when it is just by itself.

Sklearn has a built in accuracy score and F-score function. I will be using those, but these two metrics will not be enough to see how well our models are performing. The main performance metric I will be using, is the portfolio's value after the one year time frame. Then compare that with the benchmark model, of holding the stock for a year. The dollar amount at the end of the year will be compared against each other, to determine if it was better or worst. Transactions/commission rate along with long term/short term capital

gains will be taken into consideration when calculating the final value of the portfolio. If a portfolio is at a loss, it will add back 25% of the loss to the portfolio, basically claiming capital loss (assuming a 25% tax rate). I know you can only claim $3,000 loss a year, but I will not take that limit into account for this project. The best optimal model will be the one who has the highest portfolio value at the end.

II. Analysis

Data Exploration

As mentioned above, I'll be using the Open, High, Low, Close dataset per stock.

Date	Close	High	Low	Open
4/1/2014	77.378571	77.410004	76.681427	76.822861
4/2/2014	77.507141	77.639999	77.18	77.482857
4/3/2014	76.970001	77.5	76.805717	77.341431
4/4/2014	75.974289	77.14286	75.797142	77.115715
4/7/2014	74.781425	75.842857	74.555717	75.431427

Figure 1. AAPL data sample.

Above is an example of the data that I gathered for AAPL stock. Originally there is a 'Volume' column, but I have removed that since it varies too much depending on the stock. It is hard to normalize since the values can really vary a lot, per stock. I will use multiple stocks as my training set, 67 different stocks to be exact. These are the following: AAPL, GOOG, T, IMAX, IBM, NFLX, SIRI, S, PLUG, C, ZNGA, WMS, BAC, AMZN, FB, P, WM, NOK, DDD, XME, XONE, SDRL, TSLA, SSYS, TXN, F, GS, LQMT, HTZ, BAH, GLW, SPWR, BIDU, SRPT, YGE, CNX, URRE, VJET, RAD, NQ, KORS, TWTR, HLF, ORCL, WLL, BLDP, PEG, MJNA, CBIS, TM, SBUX, MBLY, MRK, DBO, PFE, CAMP, TRXC, BMY, FE, VTR, UHT, MVO, KF, RACE, STOR, MU, RTN. All this data will be combined to a single training file for the models. This will be 36,583 rows by 4 columns. It may contain some blank information or NaN but these will be removed later. It shouldn't contain any abnormalities. It's either it has the right data or its missing, basically no errors.

	Close	High	Low	Open
count	35863	35863	35863	35863
mean	57.49262574	58.18152283	56.79206723	57.50948687
std	98.58069016	99.55819521	97.53321644	98.60009562
min	0.011	0.011	0.01	0.011
25%	11.6	11.96	11.3	11.67
50%	30.75	31.25	30.27	30.790001

75%	54.279999	54.880001	53.740002	54.34
max	776.599976	789.869995	766.900024	784.5

Figure 2. Description of the entire raw input data.

Above is a statistical summary of the input data. Since the numbers are all over the place, I'll perform a few pre-processing techniques before it can be fed to the models. This will be discussed more under Data Preprocessing. Basically multiple moving averages (2, 3, 5, 10, 15, 25, 40 days) will be calculated and normalized. Since stock prices can vary a lot per stock, I will take the ratio of the current price against the previous price. This way, our prices will mostly range from 0.70 to 1.30. Fifty percent of the data will actually be between 0.96 to 1.04. Some values will be beyond this range, but the frequency of that occurring is smaller, comparatively to the rest. The range of the dates will be about 2 years, from April 2014 - May 2016. Any rows with NA will be dropped, including holidays, weekends, and the first 2 months. Since the 40-days moving average will be NA for these times. Each stocks will have about 506 rows of data. It'll have less data if the stock wasn't available from April 2014. The entire processed data is 33,117 rows by 108 columns. Twenty columns will be deleted later. In short, I started with 4 columns and will end up with 88 columns as input data to the models. What and how these 88 features are achieved, will be discussed more later.

Exploratory Visualization

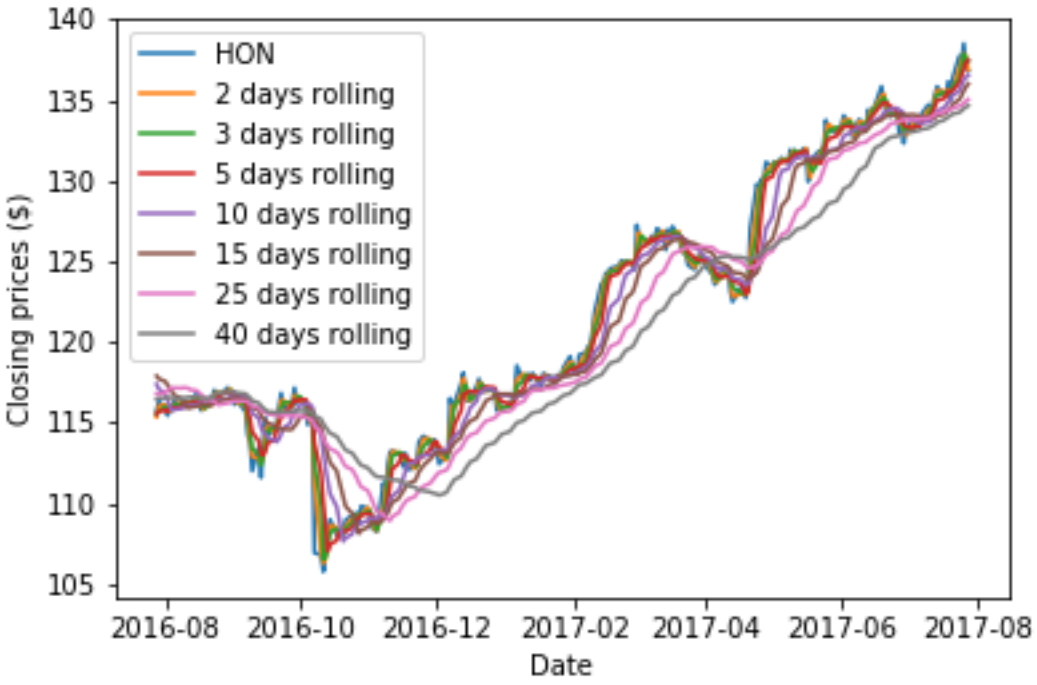

Figure 3: Honeywell Stock performance.

Above is the graph of the Honeywell stock (HON), one of our training stocks. The graph contains multiple moving averages. It is hard to follow the trend base on the current price alone. Since it jumps around with no warnings or patterns. It seems a little easier to follow the trend with the long moving average, say 40-days. The stock price is currently increasing when the slope is positive. Once it hits a negative or 0, the stock price is on the down trend. It is a few days delayed, since the portfolio value would have gone down by a bit before the 40 day moving average gets a negative slope.

Algorithms and Techniques

I'll be using Artificial Neural Networks and the following models, with default parameters:

- GaussianNB (priors=None)
- DecisionTreeClassifier (criterion='gini', splitter='best', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None, max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)
- AdaBoostClassifier (base_estimator=None, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)

I picked Naïve Bayes because it's easily trained, fast and not sensitive to irrelevant features. Although, NB assumes every feature is independent, which is not always the case. Thus it's weakness is that it can perform poorly if independence assumptions do not hold. It is still a good default model for any multi-classification problem.

I picked Decision Tree because it is easy to interpret, simple and has easy visualization. Its downside is that it can grow exponentially which requires higher memory. Another downside is that it can easily over fit the training data. This can be avoided by performing pre and post pruning, or in our case use different testing data.

Adaboost is another good algorithm because it corrects its mistakes, and it is computationally efficient. It performs bagging, which is simply choosing some subset of the training data at random, and create each instance of bags. It then performs boosting, which adds on the bagging idea where data instances that had been poorly modeled in each bags are recalculated with bigger weights or higher importance to try and correct them. Its weakness is it's not good for noisy data and needs lots of data so that weak learning requirement is satisfied. Fortunately I should have enough data, but I am not too sure if the data will be too noisy for this algorithm. I guess we'll find out.

Neural Network is easy to use, require less training, ability to detect complex nonlinear relationships between dependent and independent variables. Its weakness is that it requires tons of data. It is also like a black box that not much can be extracted from. Another downside is that it usually requires more preprocessing of the input data. All input have to be normalized or rescaled. Training target should also be one hot encoded, instead of just inputting multiple integers, like the rest of the models mentioned above. None of these are too bad, so I'll also use this model.

I originally picked Support Vector Machine (SVM) as well. Since it is good for data with tons of features. Its weakness is that a good kernel is required, which is sometimes hard to achieve. It also performs poorly on noisy data. But the reason I end up not using this algorithm, is that it takes a very long time, relative to the other models above. Without an SVM model it takes about 31 mins to complete. But once I added SVM, it takes 25 hours to generate the 120 SVM models in addition to the others. This is after I implemented multiprocessing in my code. I ended up removing the SVM models since it takes too long and it is not any better than the others.

Benchmark

I plan to compare the results with just buying and holding a stock for a certain amount of period as a baseline. For this project, I will always hold a stock for a year as a benchmark. Basically I will invest $10,000 in the beginning, per stock. Every day, the money invested will be calculated based on the change of the stock prices. The portfolio optimizer models would be executed within the same time frame, 1 year. Basically, the benchmark model will buy and hold BA, SD, FCEL, TPLM, CHK, HON, CMG, MA stocks, for a year. The portfolio optimizer models will buy/sell multiple times for a year with the same stocks. I will compare the dollar amount of the models at the end, to determine if the portfolio optimizer models are better than the benchmark model. In summary, the benchmark model's portfolio value will be proportional to the percent change of the stock price from the starting date. See figure below.

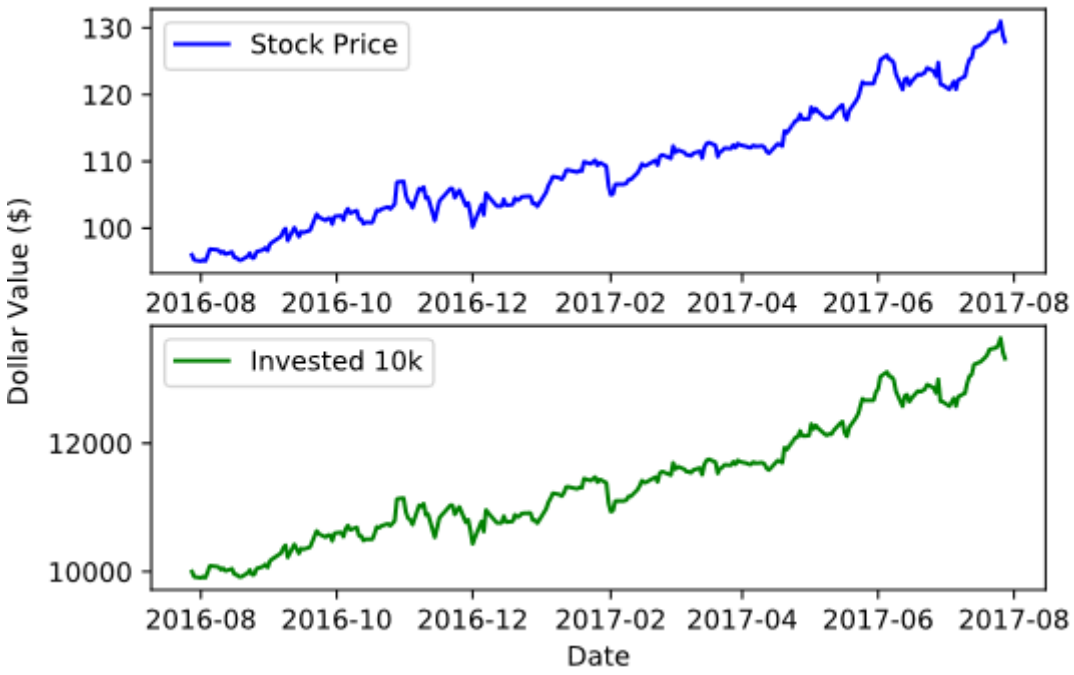

Figure 4: Example of a Benchmark model.

III. Methodology

Data Preprocessing

As mentioned above, I'll be gathering the stock data of the entire training sets. This will be 36,583 rows by 4 columns. These columns will be renamed as vOpen, vHigh, vLow, vClose. The 'v' just stands for dollar value, since others will be in ratio form. This may contain some blank or NaN, but I will handle those later. Next I will generate the moving averages. The columns vr2, vr3, vr5, vr10, vr15, vr25, vr40 will be generated. These are 2-days, 3-days, 5-days, 10-days, 15-days, 25-days, and 40-days moving averages respectively.

Next, I will generate columns prev_close, prev_r2, prev_r3, prev_r5, prev_r10, prev_r15, prev_r25 and prev_40. These are based on the previous values. Basically, a copy of the current values will be made, then shifted downwards by 1 row, so that I can get the previous values.

Next, I will generate columns Open_pc, High_pc, Low_pc, Close_pc, r2_pc, r3_pc, r5_pc, r10_pc, r15_pc, r25_pc, r40_pc. First they'll will just be a copy of vOpen, vHigh, vLow, vClose, vr2, vr3, vr5, vr10, vr15, vr25, vr40. Then I will divide all of the values by the

previous close price, hence ‘pc’. These will then become a ratio, based on the previous closing price. If the price went up, its value will be over 1.00. If the price went down, its value will be below 1.00. The columns below will then also be generated, similar to the above steps, but with respect to the other moving averages:

Open_p2, High_p2, Low_p2, Close_p2, r2_p2, r3_p2, r5_p2, r10_p2, r15_p2, r25_p2, r40_p2,

Open_p3, High_p3, Low_p3, Close_p3, r2_p3, r3_p3, r5_p3, r10_p3, r15_p3, r25_p3, r40_p3,

Open_p5, High_p5, Low_p5, Close_p5, r2_p5, r3_p5, r5_p5, r10_p5, r15_p5, r25_p5, r40_p5,

Open_p10, High_p10, Low_p10, Close_p10, r2_p10, r3_p10, r5_p10, r10_p10, r15_p10, r25_p10, r40_p10,

Open_p15, High_p15, Low_p15, Close_p15, r2_p15, r3_p15, r5_p15, r10_p15, r15_p15, r25_p15, r40_p15,

Open_p25, High_p25, Low_p25, Close_p25, r2_p25, r3_p25, r5_p25, r10_p25, r15_p25, r25_p25, r40_p25

Open_p40, High_p40, Low_p40, Close_p40, r2_p40, r3_p40, r5_p40, r10_p40, r15_p40, r25_p40, r40_p40

The above columns will be a ratio relative to the previous 2-day, 3-days, 5-days, 10-days, 15-days, 25-days and 40-days moving averages respectively.

The last column to be generated is the ‘predict’ column. I will have 108 columns at this point. The ‘predict’ column is basically the next day closing price, which will be used as the target, for our supervised learning models. This will be used to predict whether the stock is going up or down, based on the current trends. Since stock prices vary a lot, I'll be normalizing them in ratio with the previous closing price, 2-days, 3-days, 5-days, 10-days, 15-days, 25-days, or 40-days moving average prices. Basically, there will be 8 different targets. Each target will be a different model. The target ratio variable will determine which ratio the ‘predict’ column will be based on.

On top of that, I will also try multiple sell/buy prices as well. Such as:

- Below 0.985 (Sell), Above 0.985 (Buy)

- Below 0.990 (Sell), Above 0.990 (Buy)

- Below 0.995 (Sell), Above 0.995 (Buy)

- Below 1.000 (Sell), Above 1.000 (Buy)

- Below 1.005 (Sell), Above 1.005 (Buy)

- Below 1.010 (Sell), Above 1.010 (Buy)

- Below 1.015 (Sell), Above 1.015 (Buy)

- Below 0.985 (Sell), Above 1.015 (Buy), In between (Neutral)

- Below 0.990 (Sell), Above 1.010 (Buy), In between (Neutral)

- Below 0.995 (Sell), Above 1.005 (Buy), In between (Neutral)

- Below 0.995 (Sell), Above 1.000 (Buy), In between (Neutral)

- Below 1.000 (Sell), Above 1.005 (Buy), In between (Neutral)

- Below 1.005 (Sell), Above 1.010 (Buy), In between (Neutral)

- Below 1.000 (Sell), Above 1.015 (Buy), In between (Neutral)

- Below 1.000 (Sell), Above 1.010 (Buy), In between (Neutral)

The sell below and buy above is used to convert the target value. The target is a ratio based on any of the closing price, 2-day, 3-day, 5-day, 10-day, 15-day, 25-day, or 40-day moving average prices. Then the target is converted to -1, 0 or 1 based on the sell and buy ranges. If the sell is set to 1.00 and the buy is set to 1.01, then any target/predict value below 1.00 will be converted to -1 (or sell). Any number above 1.01 will be converted to 1 (or buy) and anything in between will be converted to 0. Numbers -1, 0, and 1 will be the multi-label classification for our supervised models.

All of these will be ran and I will determine which combination has the best performance. There will be a total of 480 different models, 8 (target ratios) x 15 (sell/buy prices) x 4 (algorithm models). The best optimal model will be determined base on which one has the highest portfolio value, at the end of the year time.

After the input data has been processed, all NaN or blank data will be removed. Including blank moving averages, holidays, and etc. Columns that are still based on actual stock prices will be deleted. Columns such as: vOpen, vHigh, vLow, vClose, vr2, vr3, vr5, vr10, vr15, vr25, vr40, prev_close, prev_r2, prev_r3, prev_r5, prev_r10, prev_r15, prev_r25 and prev_40. These are the 20 columns I mentioned before. The ‘predict’ column will be the

target while the rest (88 columns) will be the input for GaussianNB, DecisionTreeClassifier and Adaboost supervised models.

For the Neural Network model, I still need to do more pre-processing. First all the input data will be rescaled. Values from 0.85 to 1.15 will be rescaled to be -1 to 1. Values outside of the range will be evaluated proportionally. If the input was not rescaled or normalized, the neural network will sometimes have 0% or 90% accuracy. It will not be consistent at all. Sometimes its 0%, sometimes its 90%, which is both not accurate. Once the input is rescaled, it will be more consistent. Accuracy will then be at least 30%, but can sometimes go up to 80%. The target for the neural network also needs to be processed first. It needs to be one hot encoded before fitting it to the neural network. Instead of targets -1, 0, and 1, it'll be [1,0,0], [0,1,0], and [0,0,1] respectively.

Implementation

My supporting code is written using jupyter notebook. For reference, it can be found under: https://github.com/En3rG/MLND-Capstone

The stocks data were acquired using pandas DataReader module. Initially I was using google as the data source. This one worked fine, though some stocks seemed to be missing, I just avoided using those tickers. A few weeks during development when I was almost done, the google data source stopped working. It was no longer providing me the date range I specified. It was just giving me the past year's data, no matter what I set the date range to be. I had to use yahoo as the data source instead. First of, yahoo had an extra column 'Adj Close', which I had to delete to match my input from google srouce before. Then I initially was not getting the same results as I was with google. It turns out google returns data in Open, High, Low, Close order. While yahoo returns Close, High, Low, Open. This caused some issue with the calculations. Also, some days, the yahoo data source will sometimes not work at all. I ended up saving the training and testing data locally so I don't have to download online, for every run. Basically, bugs or API changes could definitely affect anyone's development without notice. It is good practice to save some data locally during testing or development.

The data processing mentioned above were mostly done using pandas and numpy. When manipulating some data, like getting the ratios, I mostly just used pandas. Using numpy would be faster, but the code for pandas is more readable, since you can specify the column title you want to update. For numpy, you'll just specify the column number which could be confusing, especially in the development phase. I used numpy's vectorize function to rescale the input for the neural network. Keras has a 'to_categorical()' function to one hot encode the output but I created my own encoder/decoder functions.

Each input data is saved as a csv file, for every single step. This is used for troubleshooting. Initially I wasn't doing that, and it was hard to debug the code. Whether the preprocessing is working correctly or not, or if there are any issues in between steps. Saving the raw input and the processed input, I was able to see and analyze the input and output of the preprocessing functions. I was then able to debug my code faster. The same steps were applied to the test data.

Each stock data, both the training data and test data, were also graphed and saved using matplotlib. This helped a lot in visualizing the data, and easily spotting any outliers in the data.

Here's a quick summary of the files generated by the program:

- training_stocks.pkl
    - If parameter 'READ_EXISTING_TRAINING_STOCKS' is set to False, the program will generate this file. If the parameter above is set to True, the program will read the existing file found in the local directory. This file is pandas panel data type, which will contain all the raw training stocks data.
- testing_stocks.pkl
    - If parameter 'READ_EXISTING_TESTING_STOCKS' is set to False, the program will generate this file. If the parameter above is set to True, the program will read the existing file found in the local directory. This file is pandas panel data type, which will contain all the raw testing stocks data.
- training_data/
    - Raw stock data – the raw data for each of the training stocks. Each file will have 4 columns. The number of rows will be based on the TRAINING_START_DATE and TRAINING_END_DATE
    - Processed stock data – the processed data for each of the training stocks. This will contain the 108 columns mentioned earlier.
    - All raw stock data combine – this is all the raw stock data combined into a single file.
    - All raw stock data description – this will be pandas describe() function on the 'all raw stock data' file above.
    - All processed stock data combine – this is all the processed stock data combined into a single file
    - All processed stock data description – this will be pandas describe() function on the 'all processed stock data' file above.
- testing_data/

    - Raw stock data – the raw data for each of the testing stocks. Each file will have 4 columns. The number of rows will be based on the TESTING_START_DATE and TESTING_END_DATE
    - Processed stock data - the processed data for each of the testing stocks. This will contain the 108 columns mentioned earlier.
- figures/
    - Training stocks with moving averages – Each of the training stocks will be graphed along with its 2-days, 3-days, 5-days, 10-days, 15-days, 25-days, 40-days moving averages.
    - Testing stocks with moving averages – same graph as above, but for the testing stocks.
    - Best Optimal model's performance (per testing stock) – this will be a line chart showing the stocks price performance over the year and the final value of the best optimal model (based on $10,000 initial investment).
    - Accuracy and F-score of optimal models – this will be a bar chart showing the accuracy and F-score of the optimal models for GaussianNB, Adaboost, DecisionTreeClassifier, Neural Network, and Benchmark models.
    - Overall Optimal model's performance – this will be a bar chart containing the results of all the optimal models for GaussianNB, Adaboost, DecisionTreeClassifier, Neural Network, and Benchmark models. It will show the result of each of the testing stocks, along with its final value (cumulatively).
    - Graph of all of the 120 Adaboost models, with its final value.
    - Graph of all of the 120 GaussianNB models, with its final value.
    - Graph of all of the 120 DecisionTree models, with its final value.
    - Graph of all of the 120 Neural Network models, with its final value.
- model_predictions/
    - This folder will contain all the input/output of all the models. Including the number of transactions and the money value of the model. It will have 3,840 files. 480 different models x 8 test stocks = 3,840.
- models/
    - If the parameter 'READ_EXISTING_MODELS' is set to False, it will generate and save 480 different models. Otherwise, it will read the existing models in this folder.
- temp_results/
    - temp results will contain temporary results of each of the processes. This was added when multiprocessing feature was added. All these files will be appended together for the final result_df where the optimal model will be determined.
- Results.csv

- This will contain the performance of each of the 480 models generated. Including all the transactions made, final value per stock, accuracy and F-score.

The GaussianNB, Adaboost, and DecisionTreeClassfier models all used default parameters. Updating the parameters didn't have noticeable change or improvement on the final results. The parameters such as target ratio and buy/sell price ranges had more impact on the final results. The neural network is just a simple model with 300 nodes as the first layer. The hidden layer has 150 fully connected nodes. Having a deeper network didn't have any noticeable affect, thus I kept it as simple as possible. The neural network's last layer has 3 nodes for the outputs [1,0,0], [0,1,0], [0,0,1] or -1,0,1 for sell, neutral, buy respectively. For the first two layers, the activation function used was 'tanh'. This lessens the loss compared to using 'relu' as the activation function. The last layer then uses a softmax as the activation. As for the optimizer, the stochastic gradient descent, SGD, with 0.001 learning rate was used. When the learning rate was higher than this, the accuracy and loss doesn't seem to be improving per epoch.

The data for each of the models are saved under the 'model_predictions' folder. It is a csv file with all the input used for the model, along with the model's prediction. The number of transactions and money value will also be included. The data can be analyzed for any errors in the code written. This folder will contain 3,840 files. I mentioned before that there will be 480 different models, but each stock needs to use each model. Thus 480 models x 8 test stocks = 3,840.

For example, there should be 8 different Adaboost_vr5_1.0_1.015, one for each testing stock. Adaboost is the algorithm model used. 'vr5' means that the target ratio is based on the 5-days rolling average. The sell price is below 1.0 and the buy price is above 1.015. Adding all the 8 different files (per testing stock) will result in the total value for that model. The final value will then be calculated by subtracting all transactions times the commission rate, minus any short term capital gains (25%), if any. Each transactions is $4.50, based on Fidelity's transactions fee.

As for the benchmark model, it will only subtract 2 transactions per testing stock. Which is the initial buy and the sell at the end of the one year. Long term capital gains (15%) will be deducted to the final value, if there are any.

If any model is at a loss, then 25% of the loss will be added to the portfolio, to simulate claiming a loss in investment. For this project, there will be no limit in the amount of loss you can claim. In reality, there is a $3,000 limit per year. For simplicity, I will not take that into account.

Refinement

My first run took about 50 hours on a MacBook Pro 2.5GHz i7 (2014). So the first refinement I did was improving efficiency of my code. I made sure no work was getting done twice and that my algorithms are as efficient as possible. I then implemented multiprocessing, to take advantage of multiple cores. These improvements dropped the execution time to about 20 hours. Later on I removed the SVM model, since it wasn't better than the other models anyway. This dropped execution time much further, to about 1 hour. I also used to have 7.5 years of training data. I cut it down to 2 years, that brought the execution time to about 30 minutes. If the models are being read, not generated, execution time will only be about 3 minutes.

As mentioned above, the models have default parameters. Updating the hyperparameters didn't have any noticeable effect on the results. Most improvements were achieved by having different ranges of the buy/sell parameters. Basically having different versions of inputs/outputs to our models. Thus, the GaussianNB, DecisionTreeClassifier, and Adaboost have all default parameters.

The Neural Network on the other hand have to be updated a little bit. Using tanh instead of relu as the activation function resulted into lower loss per epoch. Also, learning rate of 0.001 showed improvement on the accuracy per epoch. When it was higher, like 0.1, there wasn't any improvement on the accuracy per epoch. I also tried using deep networks, with multiple hidden layers. Unfortunately, there wasn't any significant gains compared to the very simple network. Thus, I used the simple one instead.

IV. Results

Model Evaluation and Validation

Below are the results of the entire 480 models generated. It is a little hard to see, but the svg file under 'figures' should be more readable. For the Gaussian models, majority of the best ones are the ones with vr40. Basically ones where the target are based on the ratio of the 40-days moving average. For Adaboost and Neural network models, majority of the best ones are the ones with vClose, where the target is a ratio of the previous close price. For the Decision Tree models, although the 3 best ones are from models with vClose, the majority of the best ones are from vr25 and vr40.

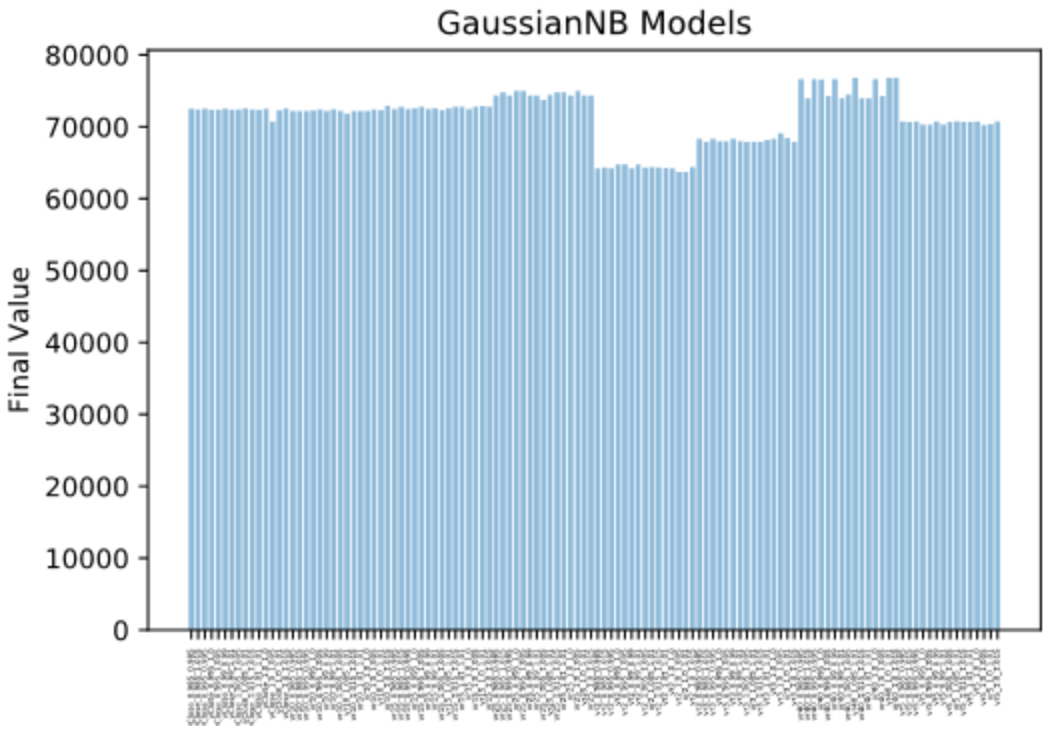

Figure 5: GaussianNB Models.

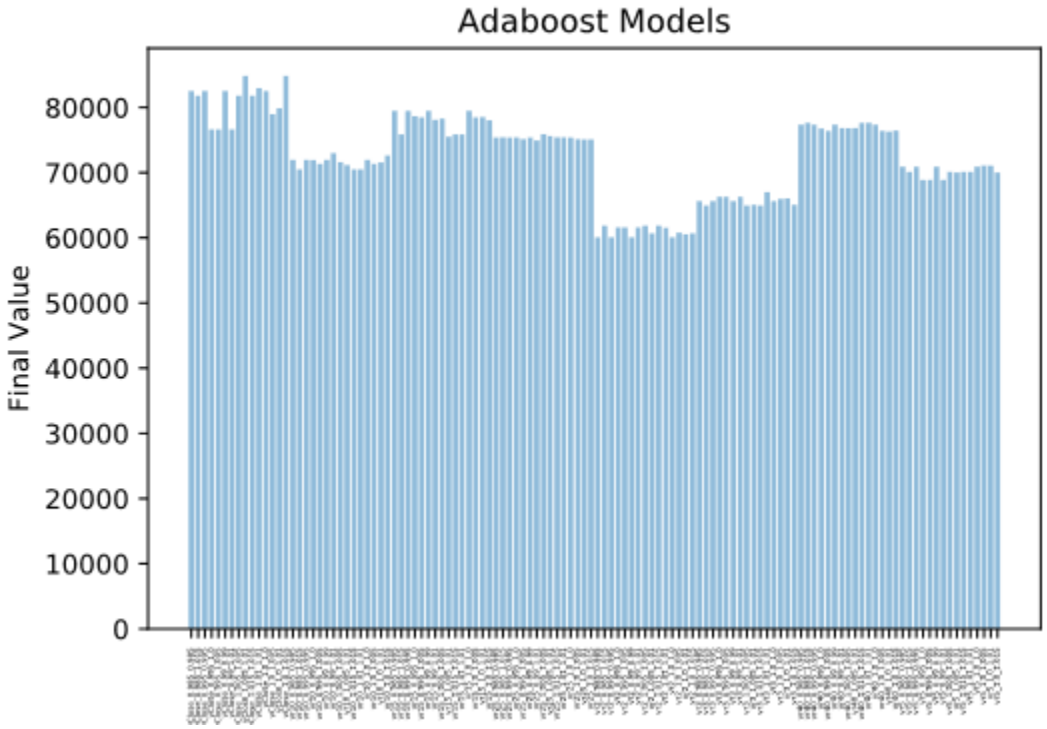

Figure 6: Adaboost Models.

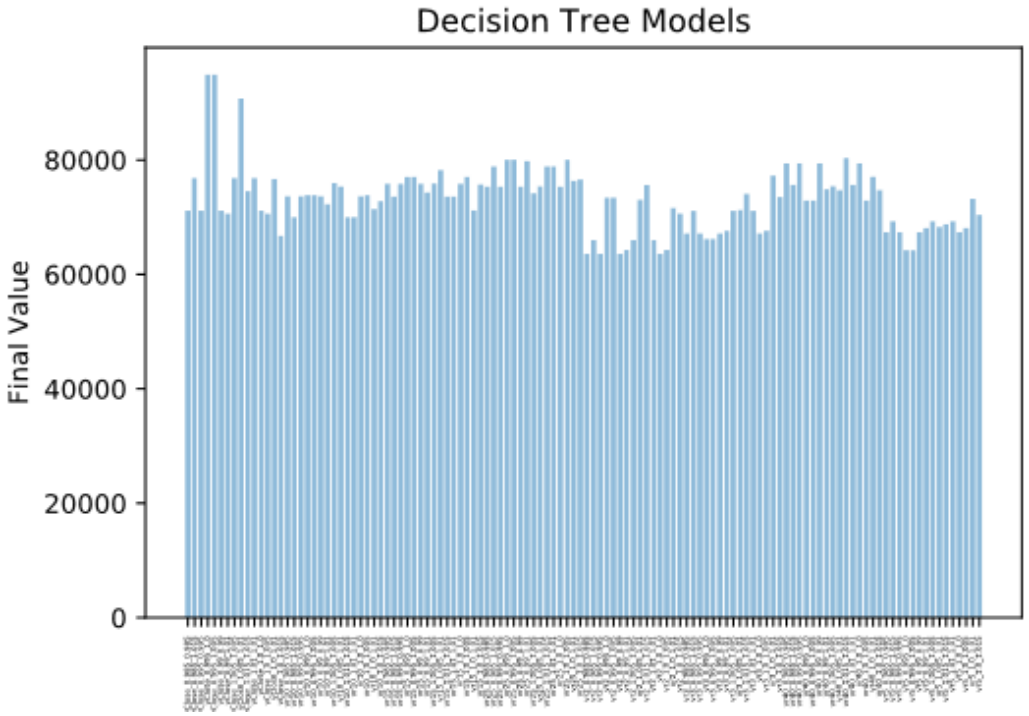

Figure 7: Decision Tree Models.

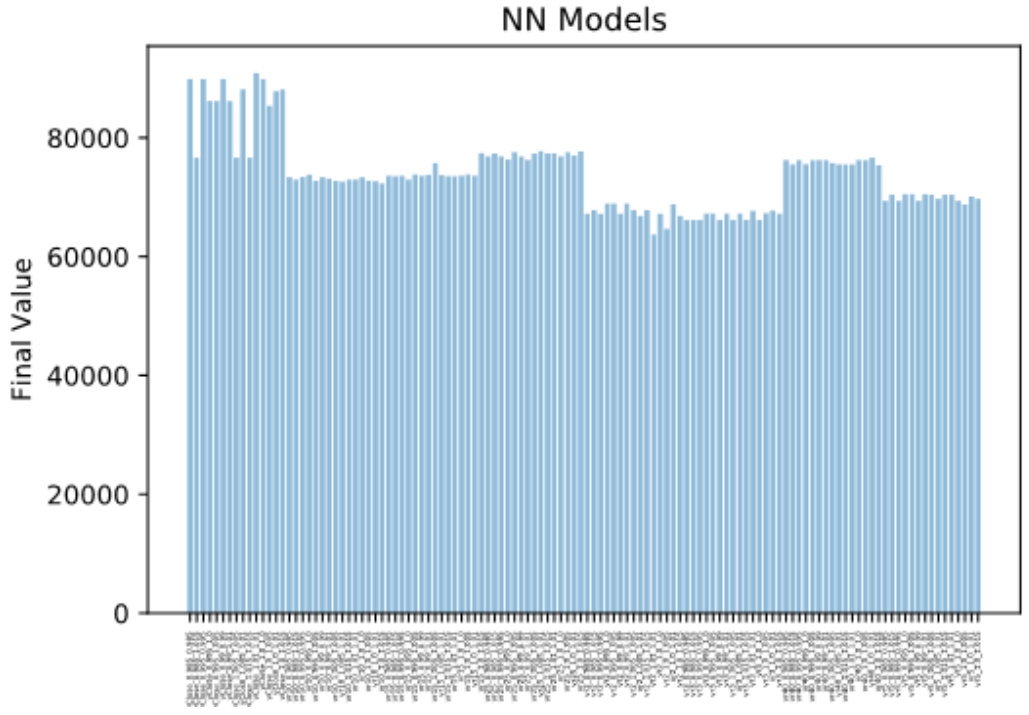

Figure 8: Neural Network Models.

Below are the accuracy and F-score of the optimal models, per algorithm. The Neural Network, GaussianNB and Adaboost have pretty low accuracy, except the DecisionTree. The decision tree usually tend to overfit the training data when not handled correctly. It is pretty hard to get a high accuracy since stocks trends are really random, or noisy. The same exact input to the model can have different results. Meaning the supervised input may state that it's a sell, while another same exact input may state that it's a buy. Thus, the models may have a hard time looking for a specific pattern.

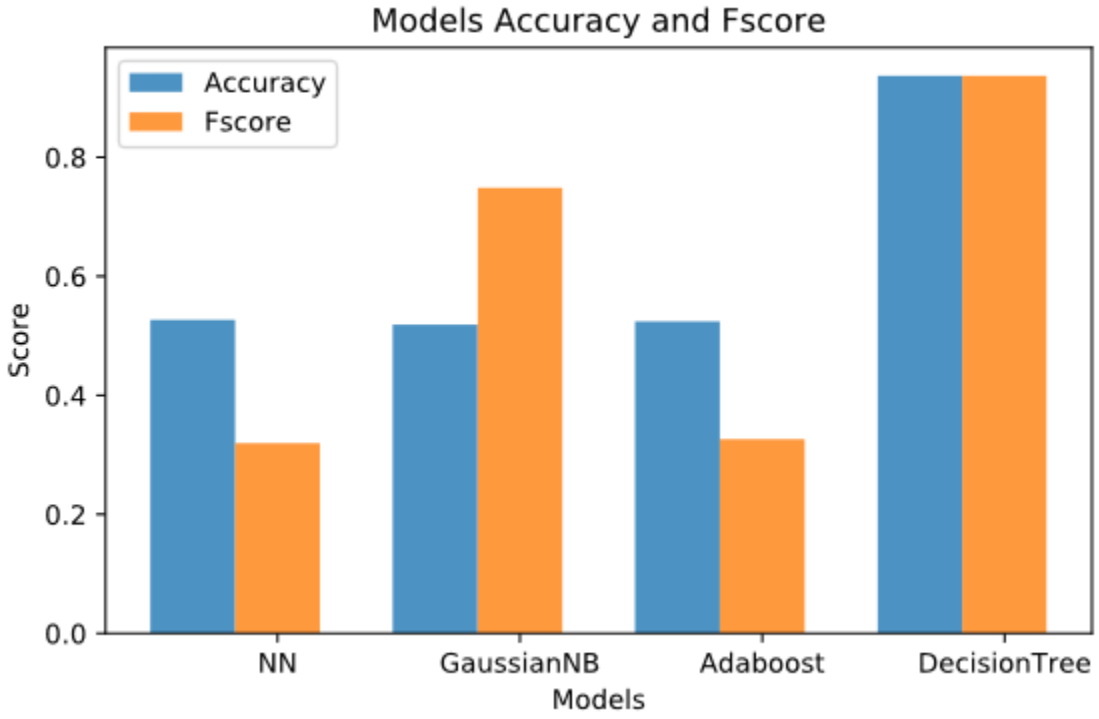

Figure 9: Accuracy and F-score of the optimal models.

Below are the historical performance of the best optimal model. The best optimal model is the Decision Tree classifier with a target ratio based on vClose. Its sell below is 0.995 and its buy above is 1.005. Thus any ratio below 0.995 was set to sell during training. And any ratio above 1.005 was set to buy.

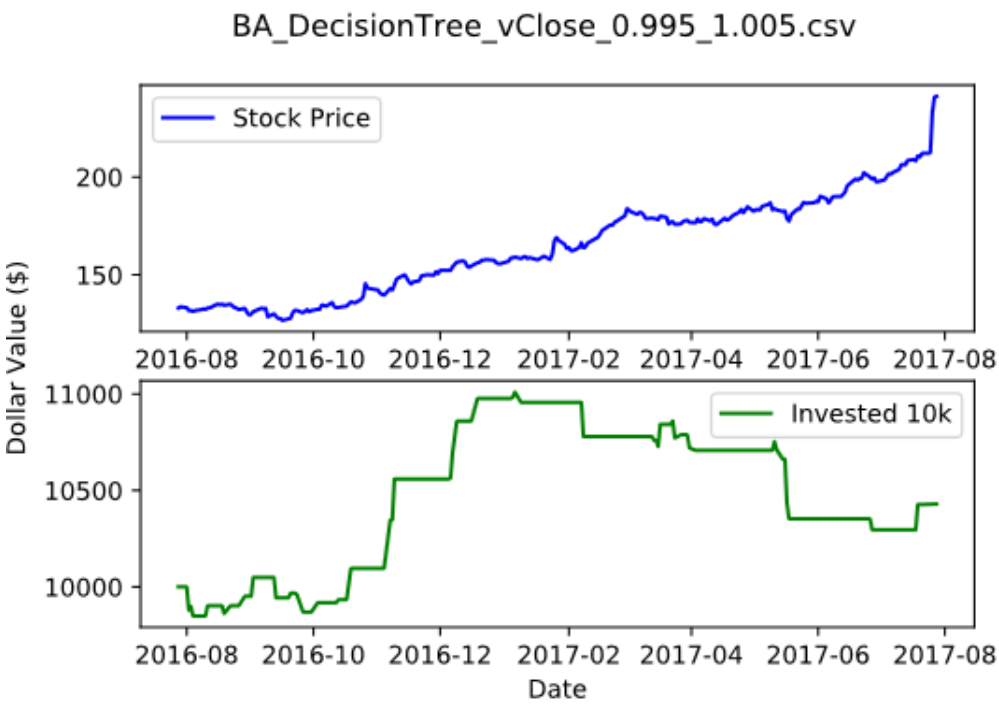

Figure 10: Best optimal model performance on BA stock.

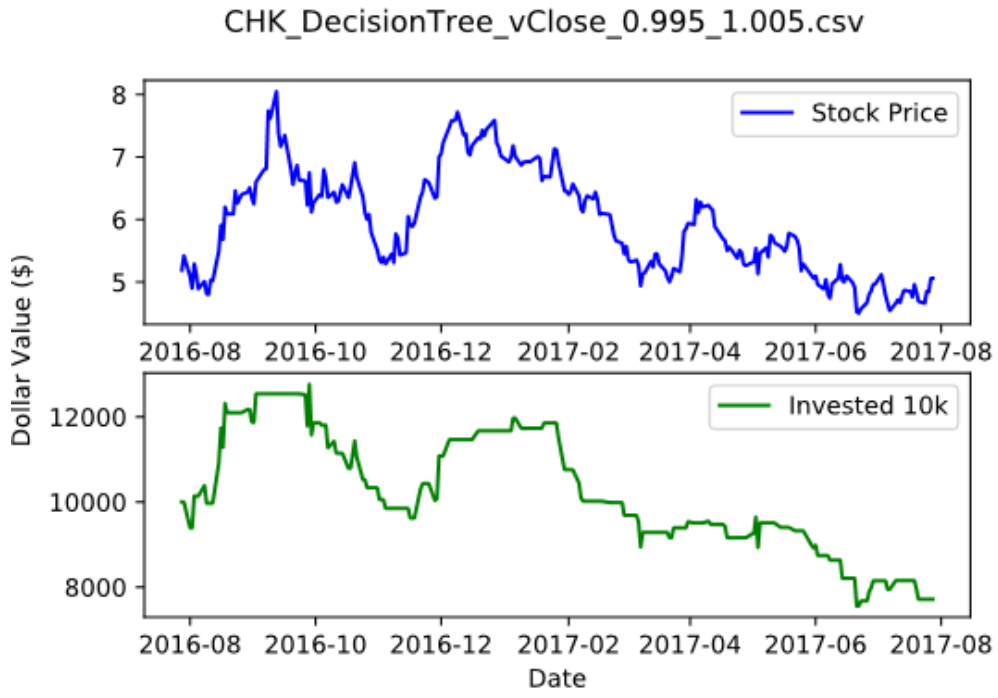

Figure 11: Best optimal model performance on CHK stock.

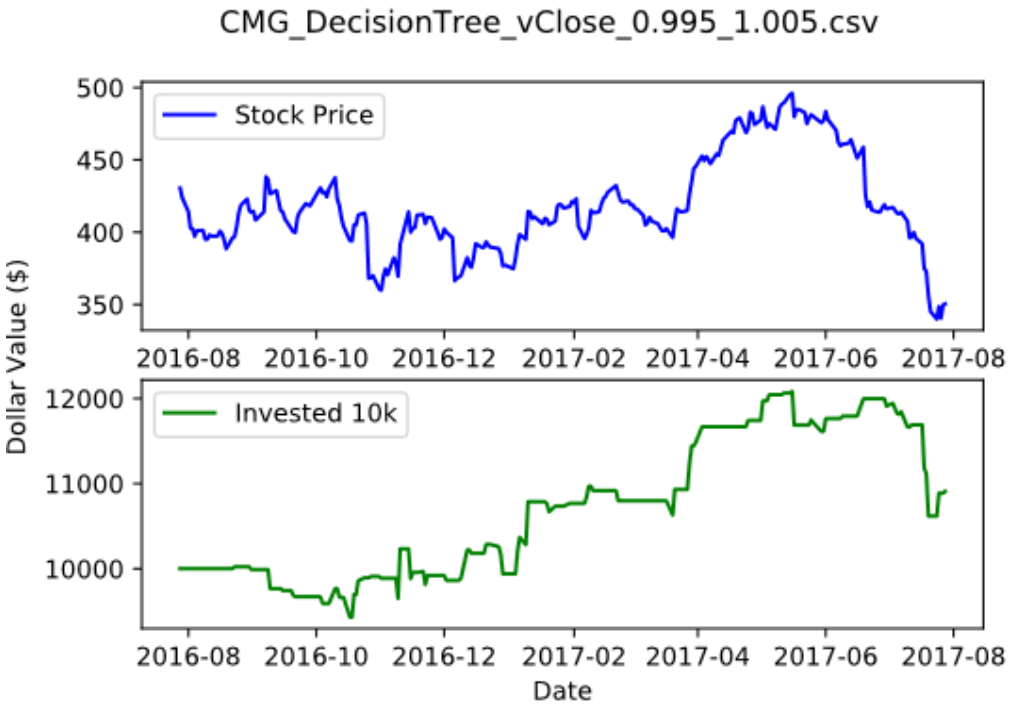

Figure 12: Best optimal model performance on CMG stock.

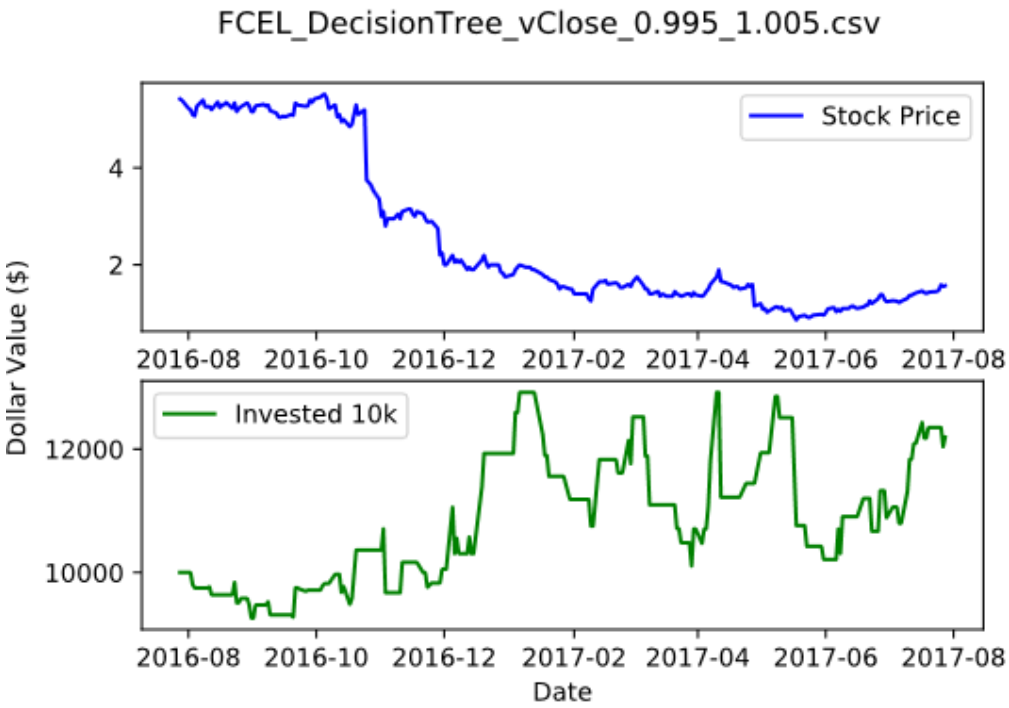

Figure 13: Best optimal model performance on FCEL stock.

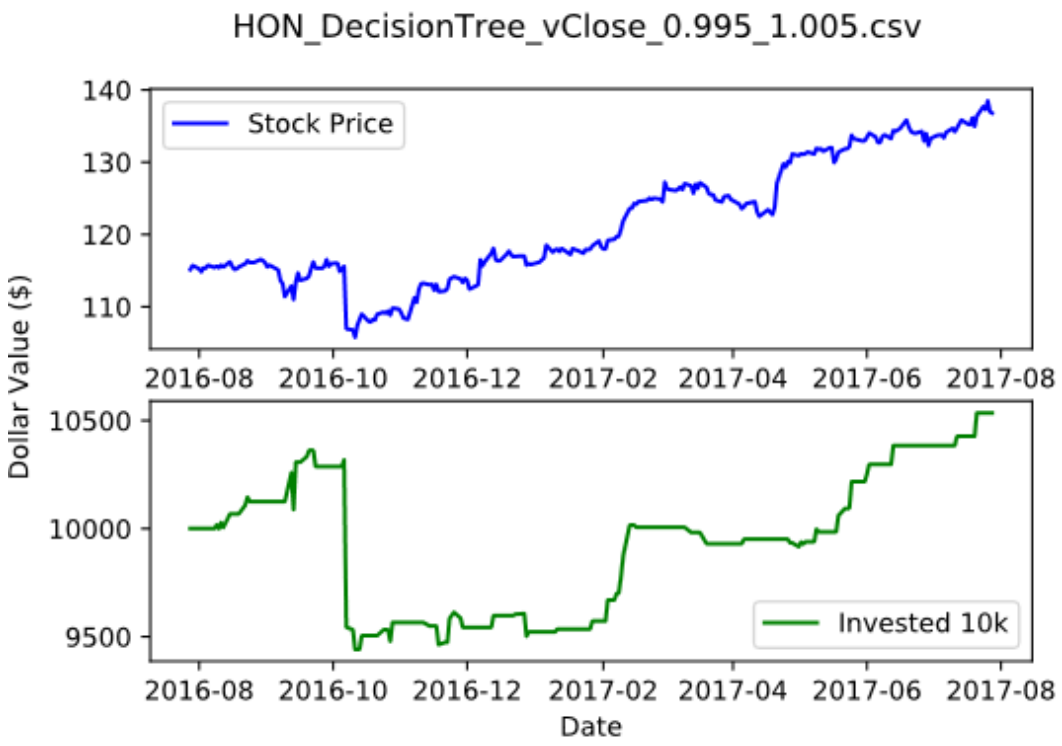

Figure 14: Best optimal model performance on HON stock.

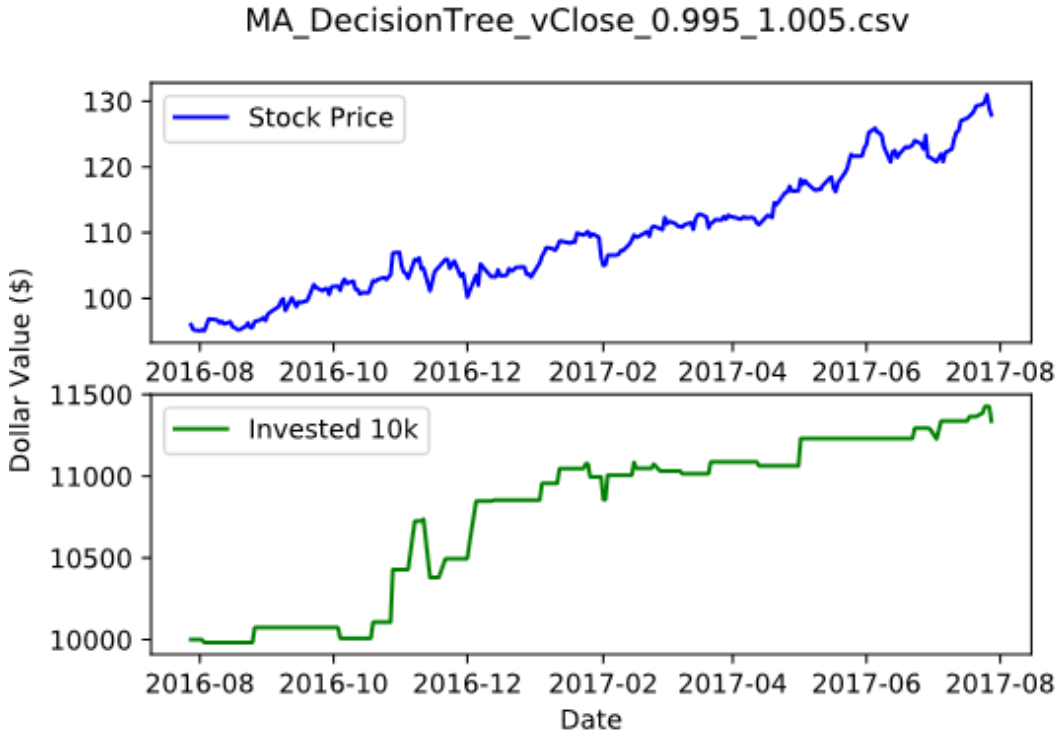

Figure 15: Best optimal model performance on MA stock.

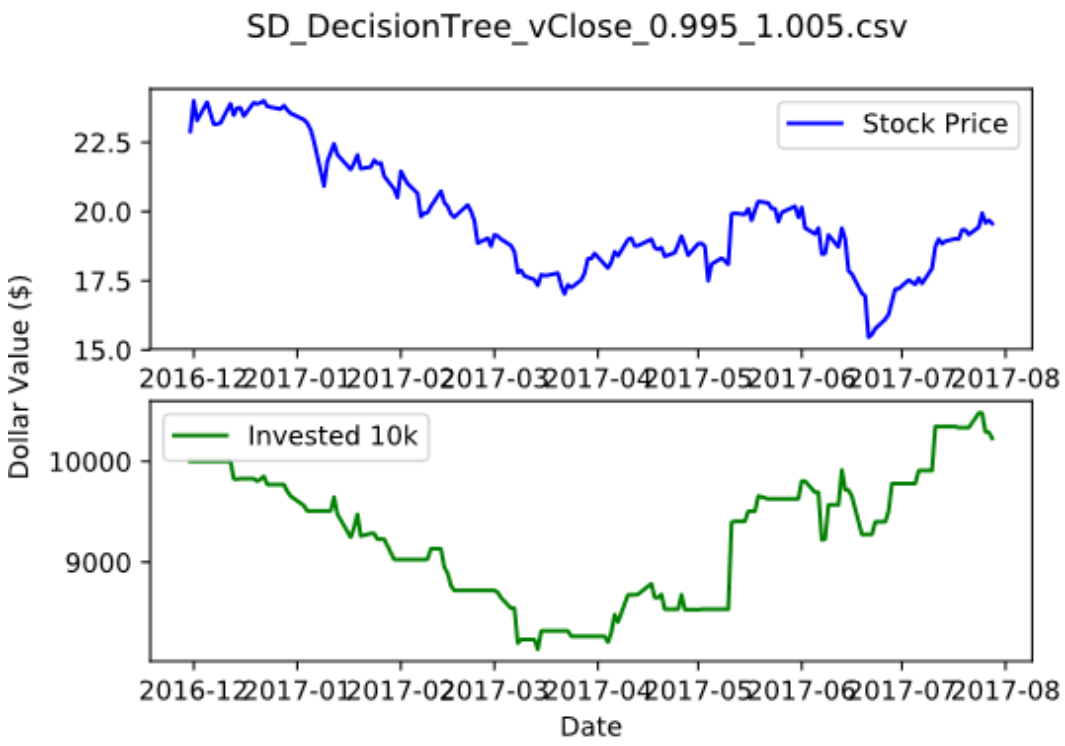

Figure 16: Best optimal model performance on SD stock.

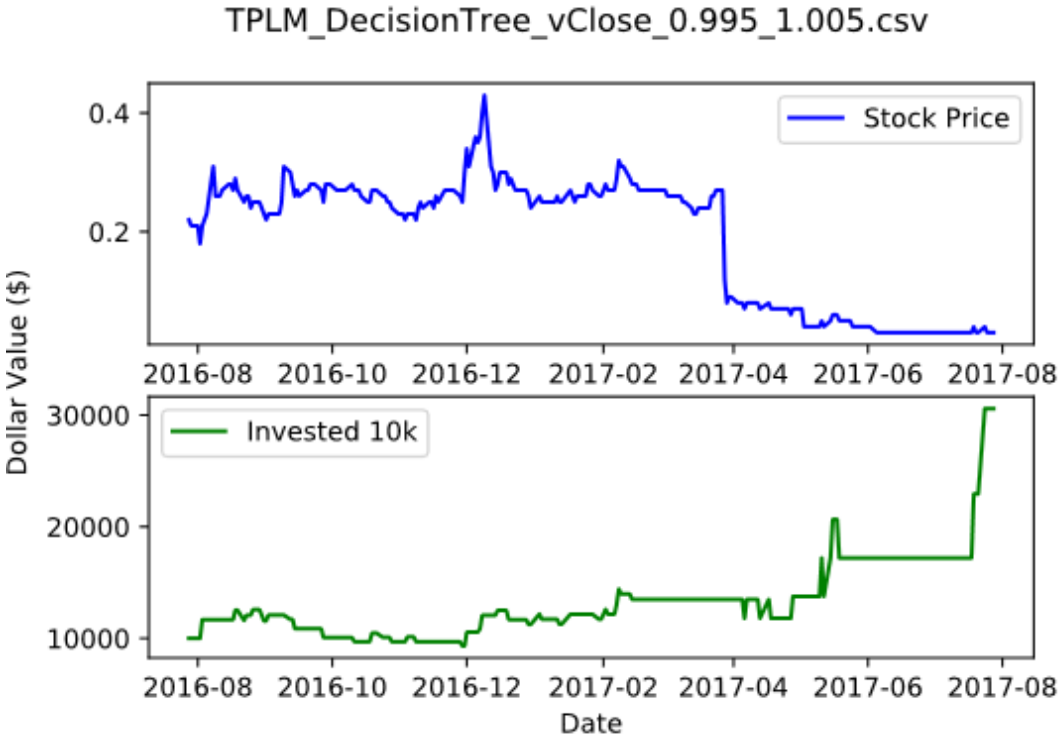

Figure 17: Best optimal model performance on TPLM stock.

Below is the summary of all the testing stocks performance for each of the optimal model. The decision tree model was able to beat the rest, especially because of its performance on the TPLM stock. It was able to generate $30,581 from its $10,000 investment. The benchmark model on the other hand had a final value of $1,363 for the same said stock. I tried using mix testing stocks, where some have an increasing trend while others have a downward trend throughout the year. This helps get a more generalized solution.

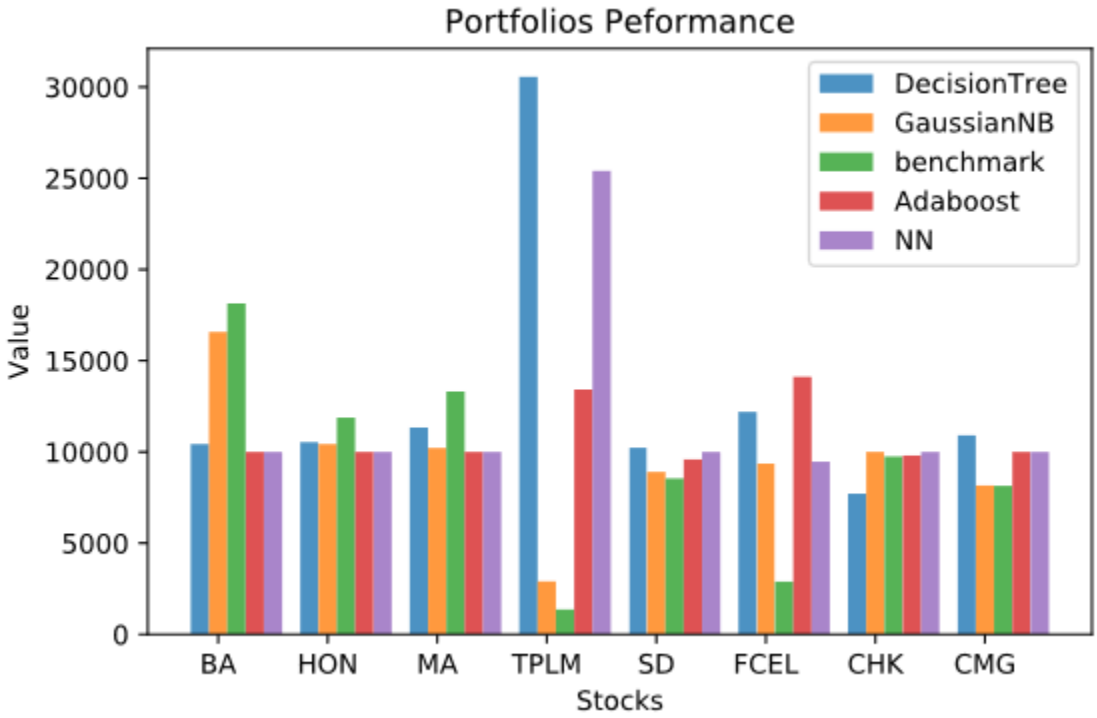

Figure 18: Optimal models performance on testing stocks.

Below is the final value of the optimal models, adding the final value of each of the testing stocks and subtracting transaction fees and long/short term capital gains. The decision tree model were able to generate $103,927 before taxes and fees. That's with the 8 stocks, each with an initial investment of $10,000. After taxes and fees, its about $94,896, almost $15,000 profit. There were 616 transactions made, for the best optimal model. The benchmark had a value of $74,037 before taxes and fees. It has 16 transactions, 8 stocks x 2 (buy/sell) for the entire year.

The neural network model on the other hand only made 56 transactions. It generated $94,868 before taxes and fees and its final value is $90,873. About $10,000 profit and still better than our benchmark model.

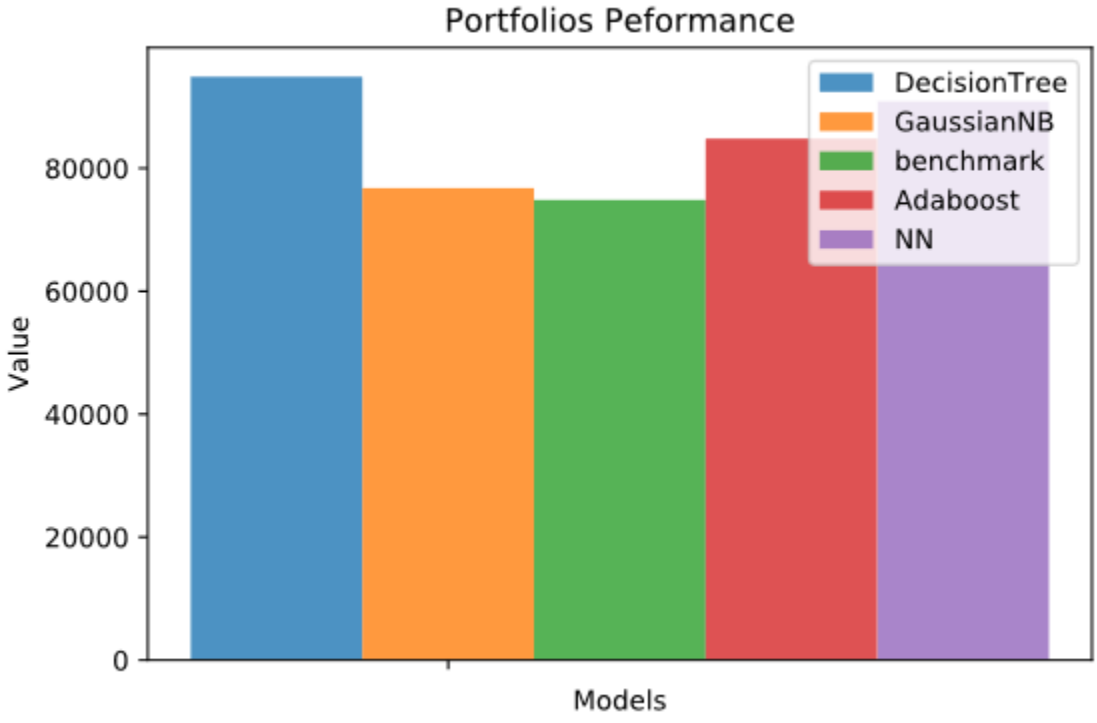

Figure 19: Optimal models final performance.

Below are the summary of each of the optimal models.

BA_benchmark	18,139.20
CHK_benchmark	9,749.50
CMG_benchmark	8,140.79
FCEL_benchmark	2,896.68
HON_benchmark	11,879.70
MA_benchmark	13,322.60
SD_benchmark	8,545.85
TPLM_benchmark	1,363.63
buy_above	0.99
sell_below	0.99
target_ratio	vr2
total_benchmark	74,037.90
benchmark_FinalValue	74,853.00

Figure 20: Benchmark Data.

BA_NN	10,000.00
BA_NN_transactions	-
CHK_NN	10,000.00
CHK_NN_transactions	-
CMG_NN	10,000.00
CMG_NN_transactions	-
FCEL_NN	9,465.17
FCEL_NN_transactions	10.00
HON_NN	10,000.00
HON_NN_transactions	-
MA_NN	10,000.00
MA_NN_transactions	-
NN Accuracy	0.53
NN Fscore	0.32
SD_NN	10,000.00
SD_NN_transactions	-
TPLM_NN	25,402.80
TPLM_NN_transactions	46.00
buy_above	1.01
sell_below	1.01
target_ratio	vClose
total_NN	94,868.00
total_NN_transactions	56.00
NN_FinalValue	90,873.80

Figure 21: Optimal Neural Network Data.

BA_GaussianNB	16,577.60
BA_GaussianNB_transactions	2.00
CHK_GaussianNB	10,001.80
CHK_GaussianNB_transactions	14.00
CMG_GaussianNB	8,169.40
CMG_GaussianNB_transactions	6.00
FCEL_GaussianNB	9,371.85
FCEL_GaussianNB_transactions	8.00
GaussianNB Accuracy	0.52
GaussianNB Fscore	0.75
HON_GaussianNB	10,430.80
HON_GaussianNB_transactions	2.00
MA_GaussianNB	10,216.50
MA_GaussianNB_transactions	2.00
SD_GaussianNB	8,904.41
SD_GaussianNB_transactions	6.00
TPLM_GaussianNB	2,934.24
TPLM_GaussianNB_transactions	20.00
buy_above	1.01
sell_below	1.00
target_ratio	vr40
total_GaussianNB	76,606.60
total_GaussianNB_transactions	60.00
GaussianNB_FinalValue	76,818.60

Figure 22: Optimal GaussianNB Data.

Adaboost Accuracy	0.52
Adaboost Fscore	0.33
BA_Adaboost	10,000.00
BA_Adaboost_transactions	-
CHK_Adaboost	9,833.75
CHK_Adaboost_transactions	14.00
CMG_Adaboost	10,000.00
CMG_Adaboost_transactions	-
FCEL_Adaboost	14,118.20
FCEL_Adaboost_transactions	36.00
HON_Adaboost	10,000.00
HON_Adaboost_transactions	-
MA_Adaboost	10,000.00
MA_Adaboost_transactions	-
SD_Adaboost	9,588.21
SD_Adaboost_transactions	2.00
TPLM_Adaboost	13,422.40
TPLM_Adaboost_transactions	26.00
buy_above	1.01
sell_below	1.01
target_ratio	vClose
total_Adaboost	86,962.60
total_Adaboost_transactions	78.00
Adaboost_FinalValue	84,835.80

Figure 23: Optimal Adaboost Data.

BA_DecisionTree	10,429.20
BA_DecisionTree_transactions	50.00
CHK_DecisionTree	7,711.18
CHK_DecisionTree_transactions	92.00
CMG_DecisionTree	10,909.20
CMG_DecisionTree_transactions	86.00
DecisionTree Accuracy	0.94
DecisionTree Fscore	0.94
FCEL_DecisionTree	12,195.00
FCEL_DecisionTree_transactions	110.00
HON_DecisionTree	10,535.70
HON_DecisionTree_transactions	62.00
MA_DecisionTree	11,337.30
MA_DecisionTree_transactions	54.00
SD_DecisionTree	10,228.70
SD_DecisionTree_transactions	68.00
TPLM_DecisionTree	30,581.00
TPLM_DecisionTree_transactions	94.00
buy_above	1.01
sell_below	1.00
target_ratio	vClose
total_DecisionTree	103,927.00
total_DecisionTree_transactions	616.00
DecisionTree_FinalValue	94,896.10

Figure 24: Optimal Decision Tree Data (Best).

The data of the 480 models can be found under Results.csv. The ones mentioned above were extracted from there. These are just the optimal models per algorithm.

Justification

The best performing model, the decision tree, is not that robust. Actually all the models are pretty sensitive. Running the program multiple times will have different results. Even though the training input are exactly the same. This is due to the randomness of the stock data. There is really not a clear pattern. Even though the decision tree classifier had the biggest return for our 8 testing stocks example, it may not perform the best when different testing stocks are used.

I don't believe the best model found in this project is significant enough to have solved the problem. I wouldn't necessarily follow the best model generated here and use it to buy and sell stocks in the real world. I don't think it's that reliable yet. Mainly because I

don't use enough information into account. Remember I only used daily Open, High, Low, and Close of the stocks. Existing models in the industry uses high frequency data, twitter feeds, news, company profits, dividends, and etc.

V. Conclusion

Free-Form Visualization

Below is a graph of the entire training data base on Open_pc, High_pc, Low_pc and Close_pc. 'Pc' stand for previous price, basically they are all a ratio based on previous closing price. If the close_pc is below 1, it is set to red for sell. When its above 1, it is set to blue for buy. The graph looks like red and blue are completely separable. This is a little misleading. This is because blue dots were rendered first, then the red dots. Overlaying it on blue, which makes it appear like it is just red dots. I graphed red and blue dots separately below to better visualize the spread. What the graph shows is that the mostly dense area is actually mixed. This shows that the input data is very noisy. This is why the models have a hard time training. Also, this is why the models are not consistent. Each run generates different results.

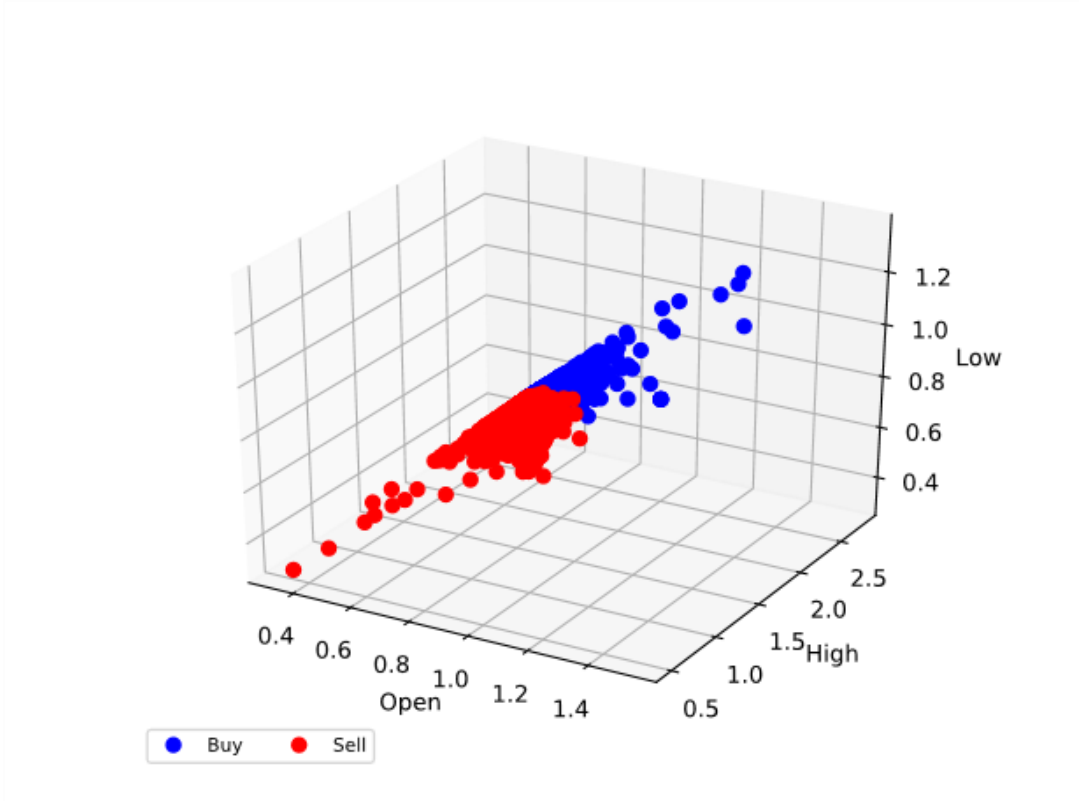

Figure 25: Close_pc graph.

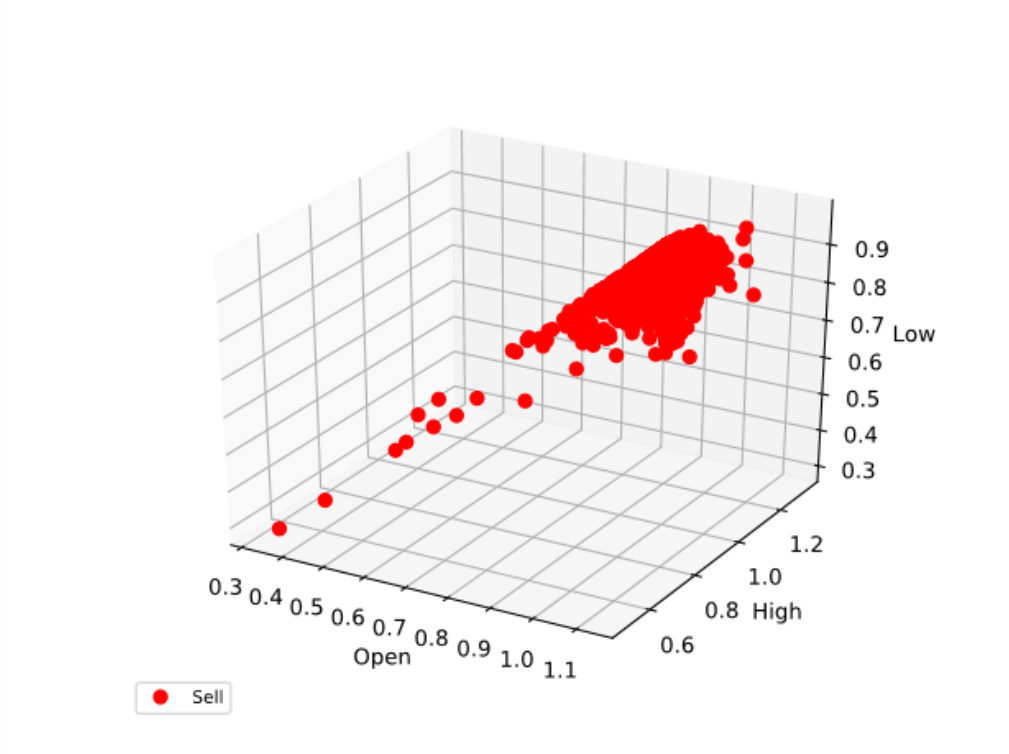

Figure 26: Close base on previous closing price (Close_pc)

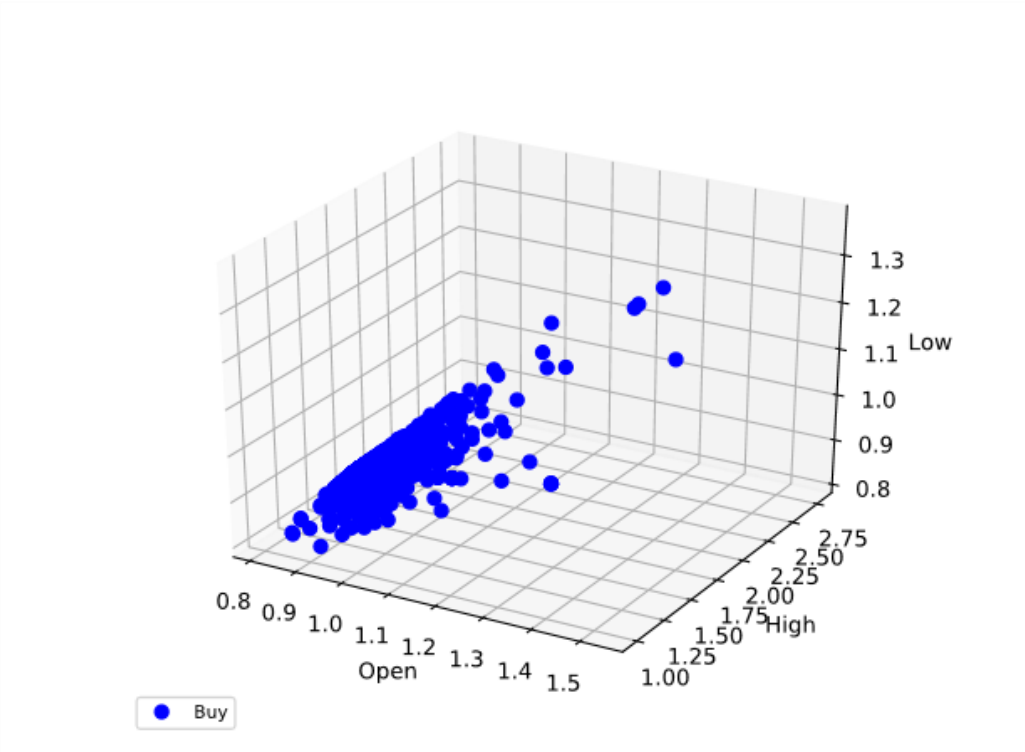

Figure 27: Close base on previous closing price (Close_pc)

Reflection

This project gave me a better understanding of the entire end-to-end process of developing models. From gathering the data, performing preprocessing, fitting the model, and tweaking parameters to get the best model. It is also very important to have a general plan or idea on how everything will work and integrate together first, before starting anything. Unforeseen changes made in between can cause potential problems or bugs with already existing code. Thus planning should be the first step and should not be skipped.

I think one of most difficult task for this project is preprocessing the input data. Once this is all done, you can just fit it into the model and update parameters. It is more of just trial and error at that point. With preprocessing, you first need to analyze the raw data you gathered. Need to extrapolate additional data if necessary. Perform calculations and make sure the code does what you actually intended to do. Possible bugs may exist, so troubleshooting and testing will take some time. Also need to take into account any requirements of the model you are using. Such as, does it need the target one hot encoded, input normalized or rescaled? Once preprocessing is complete, it is almost just plug and play.

The next task that may be challenging is generating logs and visualizations. Since the code executes multiple tasks and steps, it is very useful to have logs or visualization of each of the intermediate steps. That way any errors can be observed and pinpointed easier. My program generates over 4,500 files per run. This includes figures, graphs, tables, and models for troubleshooting and analyzing results. This seems quite a lot at first, but it is really not that much. Keep in mind there are 480 different models being generated.

Improvement

I would like to try deep reinforcement learning in the future and see what kind of results I can get from that. The problem with supervised learning is that I specify what the predictions should be, based on my understanding. With deep reinforcement learning, the model will make its own decisions, on what move is best. Of course the reward system needs to be established first, but it shouldn't be too hard since it should just be based on the monetary value of the portfolio. Sometimes deep reinforcement learning can see patterns or decisions that humans may not easily see.

I am sure that there are better solutions that exists. Especially if I train with more data. Or at a higher frequency data, instead of just daily. Per second, or even per minute data might yield better results in determining the stocks movement. It could also prevent the

portfolio in losing more money when stocks are tanking. For this project, we wait until the end of day before we perform buy/sell, which could be a little late. The downside of more data, is that it will definitely increase training time significantly.

Also better sell and buy ranges might exist, which could improve our solution. I didn't try as much ranges as I would like, since training hundreds of models is already time consuming. Especially if you don't have a dedicated computer for machine learning, preferably with good GPUs. I was only using my personal laptop. I could have made use of AWS cloud computing, but I wasn't sure how much development I was going to do and I didn't really want to spend extra.

VI. References

[1] Milosevic, Nikola. "Equity forecast: Predicting long term stock price movement using machine learning." PDF file. <https://arxiv.org/ftp/arxiv/papers/1603/1603.00751.pdf>

[2] "Keras: The Python Deep Learning Library." <https://keras.io/>

[3] "Scikit-learn Machine Learning in Python." <http://scikit-learn.org/stable/>

[4] Brownlee, Jason. "Save and Load Your Keras Deep Learning Models." June 13, 2016. < https://machinelearningmastery.com/save-load-keras-deep-learning-models/>

[5] Cisneros, Benjamin Tovar. "Accuracy versus F score: Machine Learning for the RNA Polymerases." <https://www.r-bloggers.com/accuracy-versus-f-score-machine-learning-for-the-rna-polymerases/>

[6] Machine Learning Nanodegree. Udacity. <http://udacity.com/>