Machine Learning Engineer Nanodegree

Capstone Project

Youness Assassi
February 2nd, 2018

I. Definition

Project Overview

Using machine learning to solve equity trading problems is something that I am highly interested in. According to JPMorgan[1], computer generated trades account for almost 90% of all trading volume as of February 2018. Where does this leave the small investor who does not have access to the same resources as large hedge funds and investment banks do? Resources such as computers capable of high frequency trading and expert statisticians that use quantitative analysis to beat the market.

After the market crash of 2008, I decided to get an MBA with focus on finance in order to understand why the markets fluctuate the way they do. I believed that it was important for me to gain more knowledge about the financial market so I do not make the same mistake that others made when they suffered major losses during the recession. My takeaway at the time was to not try to play the market as no one beats it in the long run. Now, with the advent of machine learning, I am not so sure that this theory holds anymore. The goal of this research is to find out if the market can be beat using some of the new theories in conjunction with machine learning and automation. This will be a breakthrough for someone like me as I may be able to invest a small amount of money and let an automated system trade on its own with the goal of maximizing profits with lower risk.

There is ample amount of research being done in this field, but much of it is not published for the obvious reason that the sponsors of the research would like to keep the information to themselves. Luckily, Udacity with the help of professor Tucker Balch of Georgia Tech has published a free course online called Machine Learning for Trading.[2] In this course, the professor explains how hedge funds use different machine learning methods to devise strategies that can beat the market. Another research paper with the same name was published by Gorden Ritter[3], a professor at NYU where he shows how machine learning,

---

[1] https://www.cnbc.com/2017/06/13/death-of-the-human-investor-just-10-percent-of-trading-is-regular-stock-picking-jpmorgan-estimates.html

[2] https://www.udacity.com/course/machine-learning-for-trading--ud501

[3] Gordon Ritter https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3015609

specifically reinforcement learning or Q learning can be applied in long term portfolio management problems.

Problem Statement

Given only the freely available historical stock market information such as adjusted stock price and volume, can a stock portfolio be actively managed for a period of time that will beat the returns of the S&P 500? The goal of this project is to build an application that can generate an optimized portfolio of stocks given a certain amount of money, then adjust the portfolio as the market changes. The portfolio returns will have to beat the market returns within a defined time period of no more than 1 day in the future through a simulation using real market data. This is regression type of a problem, where I will use the S&P500 statistics as input and individual stock price as output. Some of the input variables I will be trying out will be the Bollinger Bands®, momentum and daily average.

Datasets and Inputs

I will be using the S&P 500 historical stock prices[4] that includes the high, low, open, close and adjusted close of each stock. Out of the 505 stocks, my application will generate a portfolio of about 10 stocks. This portfolio will be optimized for its high Sharpe ratio value with a percentage allocation for each of the stocks. The idea is to then analyze these stocks by generating a number of statistics that can be used as input variables or features. Some of these statistics include the Bollinger Bands, momentum and moving average, while the future stock price will be the label or output variable. The model will be trained using a number of days in the past with a target price of 1 day in the future. I will be using TimeSeriesSplit from Sklearn to split the data between training and testing. This will ensure the proper evaluation of the model as it will not get access to future data.

The goal is to eventually be able to train the model with the most up-to-date information on stock performance so that it can be able to generate recommendations around necessary adjustments to the portfolio in order to maximize its return.

According to Tucker Balch[5], technical analysis can be used to build a short term strategy (hours to a handful of days) that is able to beat the market which I plan to use here. The opposing strategy called fundamental analysis would be more appropriate for a long term strategy (months to years).

Solution Statement

[4] https://github.com/younessassassi/machine-learning-udacity-capstone/tree/master/data/stock_dfs

[5] https://www.cc.gatech.edu/~tucker/

The plan is to start with a portfolio of the entire S&P 500, then based on the historical performance of the stocks that make up the S&P 500, generate a portfolio of about 10 stocks that is optimized using the Sharpe ratio. This would ensure that we have a portfolio with high return and low volatility. The next step is to generate other statistics for each of these stocks, such as the Bollinger Bands®, momentum and Moving Average that can be used as features for our learning algorithm. The label of course will be the stock price 1 day in the future.

The plan is to try different algorithms to generate the model, including Linear Regression, KNN and Random Forests. Once the model is trained with the features from each of the stocks, I will use the model to predict the portfolio price 1 day into the future and then compare it with the actual price. The total portfolio value will then be compared to the value of S&P index.

Metrics

The benchmark model that I will be using is the performance of the S&P 500 during the same period used for the generated portfolio. Since models built based on technical analysis do not perform well in the long term, the comparison will be limited to 1 day in the future.

After generating the portfolio of stocks, the application will perform a comparison of the generated portfolio with the returns of the S&P 500 in the same period. The formula used will be:

$$E = \frac{P1 - P0}{P0}$$

P1 = Total portfolio value at beginning of testing period

P0 = Total portfolio value at end of testing period

The application will also utilize the Root Mean Squared Error of Predictions as a metric to evaluate the performance of each of the models given that this is a regression problem.

$$RMSErrors = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y_i)^2}{n}}$$

y_i is the actual stock price value

$\hat{y}_i$ is the predicted value

n is the number of test data rows

II. Analysis

Data Exploration

The data used for this project was retrieved from yahoo finance. The module 'processing.start' handles the automatic retrieval all of S&P 500 historical stock information since 2006. New data can be retrieved by typing 'python –m processing.start' in the command line from the root of the project.

The stock daily information initially includes the following parameters:

- Date: the date the was traded on
- Open: The price of the stock at the market open
- High: The highest value the stock reached that day
- Low: The lowest value the stock reached that day
- Close: The price of the stock at the market close
- Adj Close: The closing price with consideration of other factors such as stock splits and dividends.
- Volume: The trading volume of the stock on that day.

In order to achieve the first goal of finding an optimal portfolio based on historical data, the first step the application performs is to collect The Date and Adjusted close values from all of the stocks that make up the S&P 500. Unfortunately, not all of the tickers will have a complete set of data from the desired start date. For that reason, we apply a number of techniques to fill the missing data as much as possible. The forward fill methodology is the first one I used so that the last known value is available at every time point. If the last known value is not available then I applied the backward fill methodology, only this time we borrow the value from the future incase forward fill fails. If any cells are still left empty then we drop them at this point.

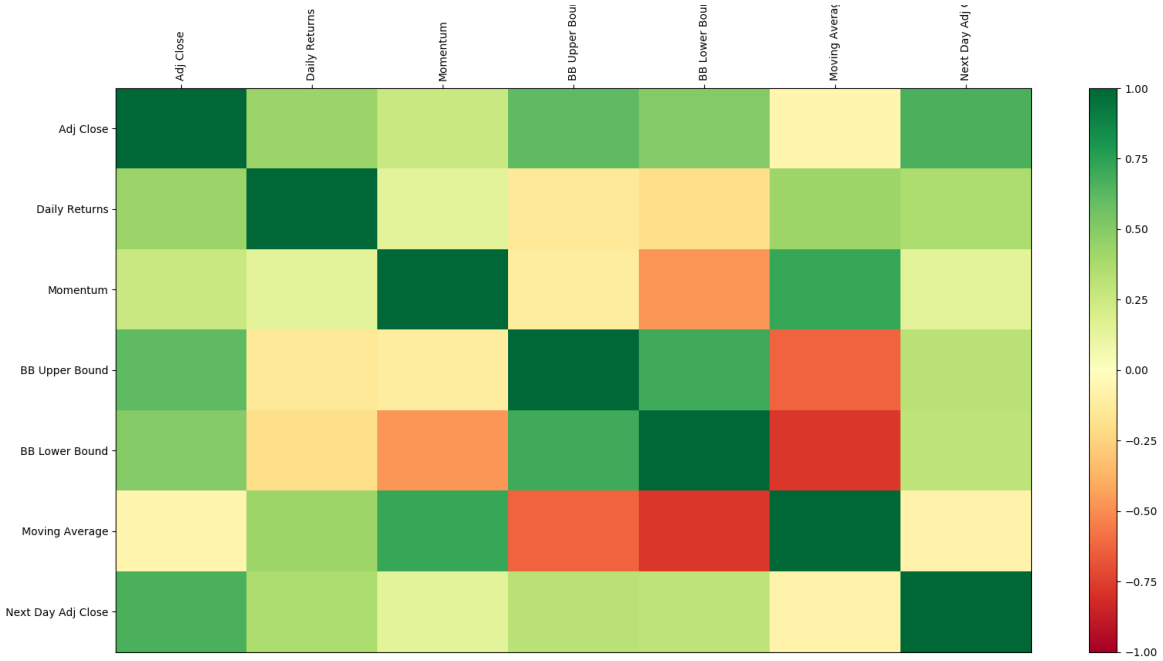
The next step is to generate the optimal portfolio combination of stocks. For this step, I used the entire set of S&P stocks as input, then applied scipy.optimize which tries to reduce the error by finding the highest Sharpe ratio of the stock combination.

$$\text{Sharpe ratio} = \frac{\text{Mean portfolio return} - \text{Risk free rate}}{\text{Standard deviation of portfolio return}}$$

The next step is to apply machine learning to try and predict the value of the suggested portfolio. This would allow us to decide whether we should invest in the portfolio or not. The data I used as features for machine learning algorithm includes:

- Bollinger Bands®: two standard deviation away from the stock simple moving average
- Momentum: the rate of acceleration of a stock's price.
- 5 day Simple Moving Average
- Daily Returns

Exploratory Visualization



PGR feature Correlation

The figure above is a representation of the correlation between the different features for the PGR Stock Symbol data for year 2017. The adjusted close and next day adjusted close prices are left here for display only and will not be used as features to train the regression models.

As can be seen on the heatmap figure, there is a strong negative correlation between the Simple Moving average and the lower Bollinger Band®. There is also a high correlation between Momentum and the Simple Moving average. I will most likely be removing some of these highly correlated features from the model training as they will only introduce noise and complexity to the model.

Algorithms and Techniques

The first technique that I will be using is to generate a portfolio of stocks with the goal of maximum return at a lower risk. To do that I will first generate a portfolio composed of all 505 equally weighted stocks from the S&P 500. Then use an optimizer called scipy.optimize that will try to reduce the error in the portfolio by maximizing its Sharpe ratio. I will be using the following parameters for the minimize function of scipy.optimize:

- Error function that optimizes for the Sharpe ratio
- Method name: SLSQ
- Bounds: between 0 and 1 for weights so that we do not end up with more than 100% allocation for a single stock

What we end up with is a portfolio of 10 stocks or less, with various weights allocated to each one of these stocks. The next step is to predict the future price of each of stock from the new portfolio. This will help determine when it would make sense to buy and sell the individual equities making up the portfolio. For this step, I will be training and testing the models using various regression algorithms with different parameters including:

- Linear Regression
    - No parameter setting
- Support Vector Machines Regression
    - Kernel: rbf, poly and linear
    - C: 1e3
    - Gamma: 0.1
    - Degree: various numbers
- Nearest Neighbor Regressor
    - n_neighbors: 3, 5, 7
- Random Forest Regressor:
    - Max_depth: various numbers
    - Random_state: 0 to ensure we get the same result every time.

The Support Vector Machine algorithms perform poorly when trained against the original distribution of the data. For that, I will be using StandardScaler to preprocess the data so that it is well normalized (values between 0 and 1 instead of the actual values).

Also, since this is a time series regression problem, I will have an issue when splitting the data for testing and training using the standard cross validation techniques. Instead, I will be using TimeSeriesSplit which always provides testing data that the model has never been trained on. This will ensure that the results we get are not tainted by the fact that the model was able to peak into the future.

Benchmark

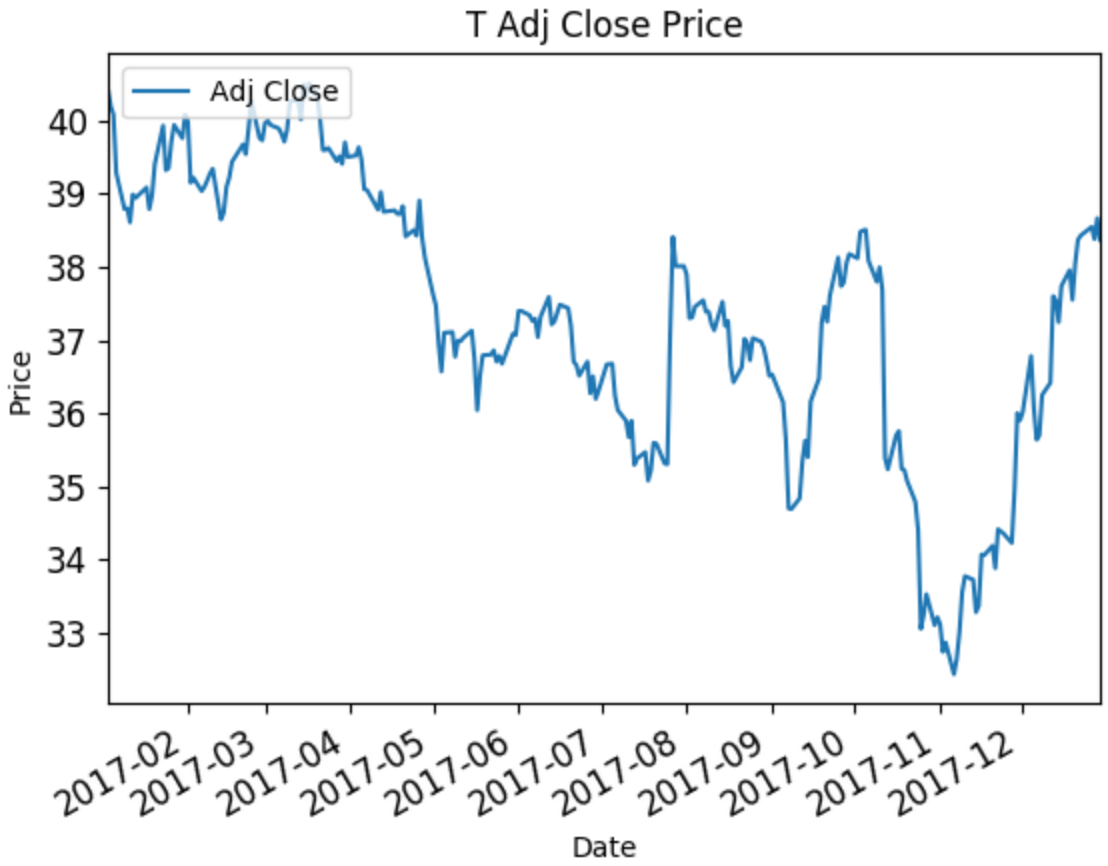
The benchmark model that I will be using is the performance of the S&P 500 during the same period used for the generated portfolio. Since models built based on technical analysis do not perform well in the long term, the comparison will be limited to 1 day in the future. I will also use the holding of the portfolio over the same period of time as a secondary a benchmark to see if the selected model performs better or worse.

The formula used will be: (P1 – P0)/P0 where P1 represents the total value at the end of the test period and P0 represents the total value at the beginning of the testing period. I will also use the Root Mean Squared Error of Predictions as a metric to evaluate the performance of each model. The model with the lowest RMSE will be the one I use for predicting the portfolio value.

III. Methodology

Data Preprocessing

After scraping the stock information from Yahoo Finance, the next step was choosing the relevant information. The downloaded data from yahoo finance includes the following columns:



Only the Adjusted Close and Date will be necessary from this data set for the next step which is to generate the rest of the features we will be using. One of the issues I faced here is that some of the stocks either did not exist from the day we need the information, or for some reason had values during the weekend when the stock market was closed. Before we do any calculation, we need to make sure we have the correct the dataset.

By using Pandas DataFrame, it is easy to fill out empty cells with relevant information or drop certain rows completely. In this case, we combine the full set of stocks information in a DataFrame and then add S&P index data. The next step is to drop the DataFrame rows where we have empty S&P cells. So that solves the issue with stocks with values that do not correspond to the S&P trading dates. We still have other stocks that did not trade at all in certain periods, and for those we use a forward fill methodology in which we fill the adjusted close values of a certain stock with the last known value. Now if those stocks do not have any last known value, we use a backward fill methodology where we use the first value we have to fill the past dates with missing information. If for any reason we still have empty cells, then we just drop the rows entirely.

The next step is to generate the various features we need. I created Ticker and TickerAnalysed classes for this reason. Ticker is responsible for collecting the ticker data and cleaning it up, while TickerAnalysed which is a sub-class of Ticker is responsible for calculating the different

statistics and that includes Daily Returns, Momentum, Simple Moving Average, Upper Bollinger Band® and Lower Bollinger Band®.

Because of the high correlation between some of the features described above, I will most likely be dropping some of them if I do not see any performance update. Also, since some of the regression algorithms, namely the Support Vector Machine Regression algorithms, require data to be normalized, so that will be done right before we run the data as normalization does not lead better results from other algorithms like K Nearest Neighbor and Linear Regression.

Implementation

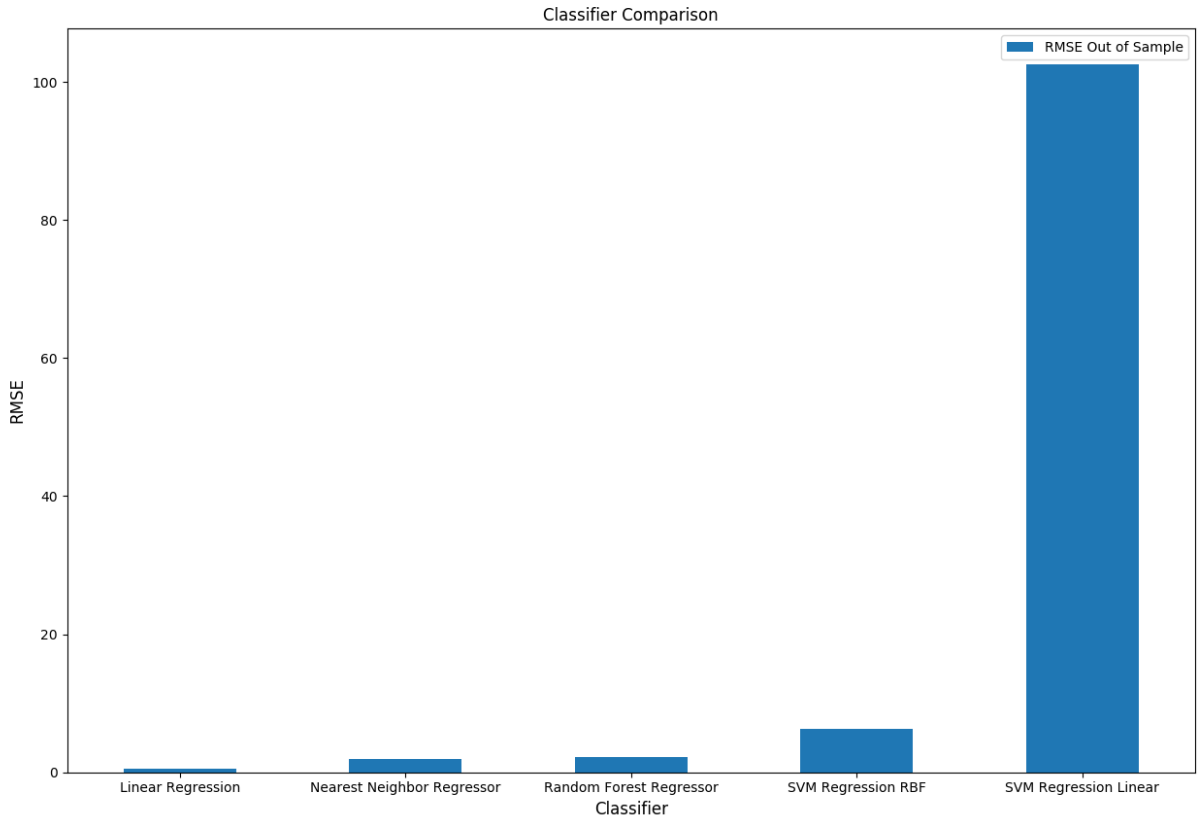
Now that I have collected and refined the features for the stocks that make up the optimized portfolio, the next step is to start training the models using the various algorithms I had selected for this project. To facilitate the comparison between the different models, I chose to create a dictionary that contains the classifiers along with their corresponding names. This enabled me to easily train, test and then compare the results of each of them. The result is then stored in a csv file called 'data/,'ticker_stats/classifiers_resuls.csv' I also created a list of algorithms that required the normalizing of training data.

The next step was to split data for training and testing. The features I used correspond to each of the individual stocks that make up the optimized portfolio. The plan is to train each of the models individually against the portfolio stocks. Since we are dealing with time series data, I used TimeSeriesSplit to split the data into 3 different train test groups. I checked the classifier against the list of classifiers that require training data to be normalized, and if a match is found then I ran the data scaling process at that point.

I then started running each of the data splits into a function that generated a confidence score and predictions. I then used those predictions to calculate the Root Mean Squared Error for both in sample of out of sample data. The generated metrics are then averaged out and stored for each of the classifiers for comparison.

Refinement

The initial results from my classifier comparison is as follows:



This above graph represents RMSE error comparison between various classifiers. Linear Regression seems to perform better than the rest of the classifiers.

SYMBOL	CLASSIFIER	CONFIDENCE	RMSE IN SAMPLE	RMSE OUT OF SAMPLE
PGR	Linear Regression	0.896780382	0.421814042	0.426065576
TPR	Linear Regression	0.713578196	0.918285866	0.902138339
DPS	Linear Regression	0.788429498	0.968281173	0.952471044
CCI	Linear Regression	0.759542548	1.19140107	1.125052406
DPS	Random Forest Regressor	0.541547245	0.480920738	1.366738715
DPS	Nearest Neighbor Regressor	0.418085838	0.986560042	1.51502509
PGR	Nearest Neighbor Regressor	-0.936745411	0.434620597	1.854789349
PGR	Random Forest Regressor	-1.028552194	0.246945529	1.873202627
STZ	Linear Regression	0.913736301	2.77617494	1.928677022
TPR	Nearest Neighbor Regressor	-0.636224249	0.877047536	2.390318276

The table above represents the top 10 classifier results per symbol.

When comparing the Root Mean Squared Error (RMSE) for each of the models and the different stocks, we can clearly see that Linear Regression is easily outperforming the rest of them. As you can see in the table below, we compare results for SVM Regressor using different parameters. SVM Regressor using RBF kernel seems to perform the best in my test but it still not performing at the level of the simpler Linear Regression classifier.

Symbol	Classifier	Confidence	RMSE In Sample	RMSE Out of Sample
CCI	SVM Regression RBF	-18.46734788	6.937406914	8.230291992
CCI	SVM Regression Linear	-18.21656303	583.1825092	417.3561826
CCI	SVM Regression Poly	-49.28776735	26205.25576	21299.18576

IV. Results

Model Evaluation and Validation

Using training data between 2017-01-02 and 2017-12-01 and all of the features available, the profile optimizer generated the following set of symbols and their corresponding weights:

NEE	CBOE	SBAC	PHM	MCD	AVY	PGR	ABBV	BA	ISRG
19%	16%	10%	10%	10%	8%	8%	7%	7%	6%

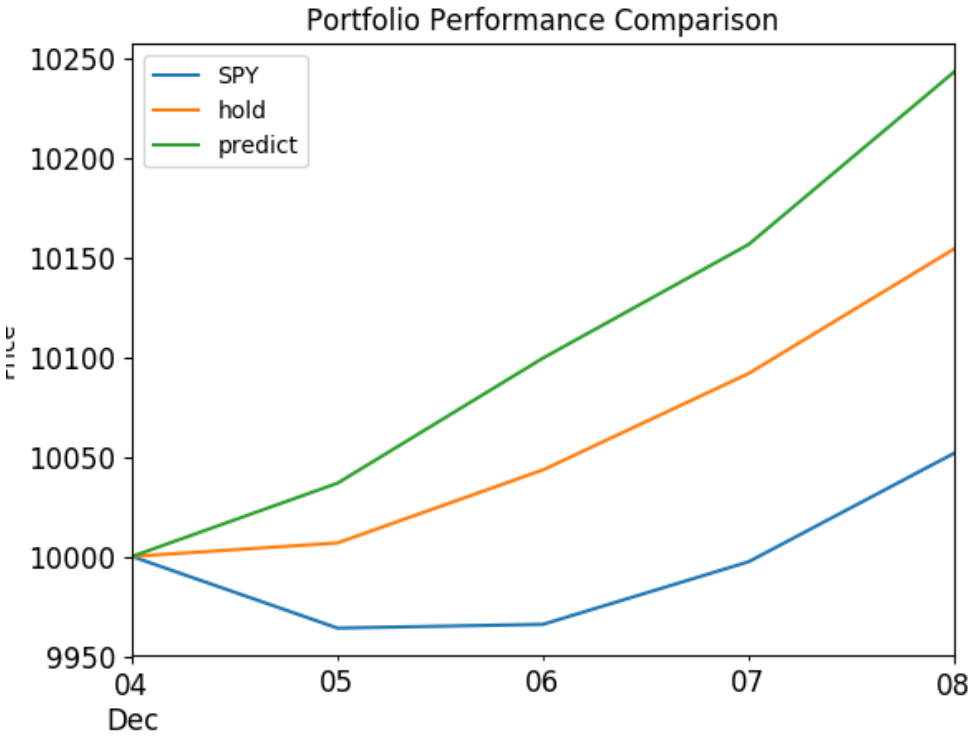
Features Used:

- Bollinger Bands®: two standard deviation away from the stock simple moving average
- Momentum: the rate of acceleration of a stock's price.
- 5 day Simple Moving Average
- Daily Returns

Result:

The following are the model results when used to actively trade the optimized portfolio between 2017-12-04 and 2017-12-08. Those results are also compared to both the performance of holding the portfolio during the same period and holding the S&P 500 stock index during the same period. The results clearly show that the active trading strategy is beating the other strategies in the span of 1 week with 11 months of training data.

S&P	Hold strategy	Active trading Strategy
0.52%	1.35%	2.39%



Performance of three trading strategies compared

Next we will train a new Linear Regression model for the same period of time but with fewer features:

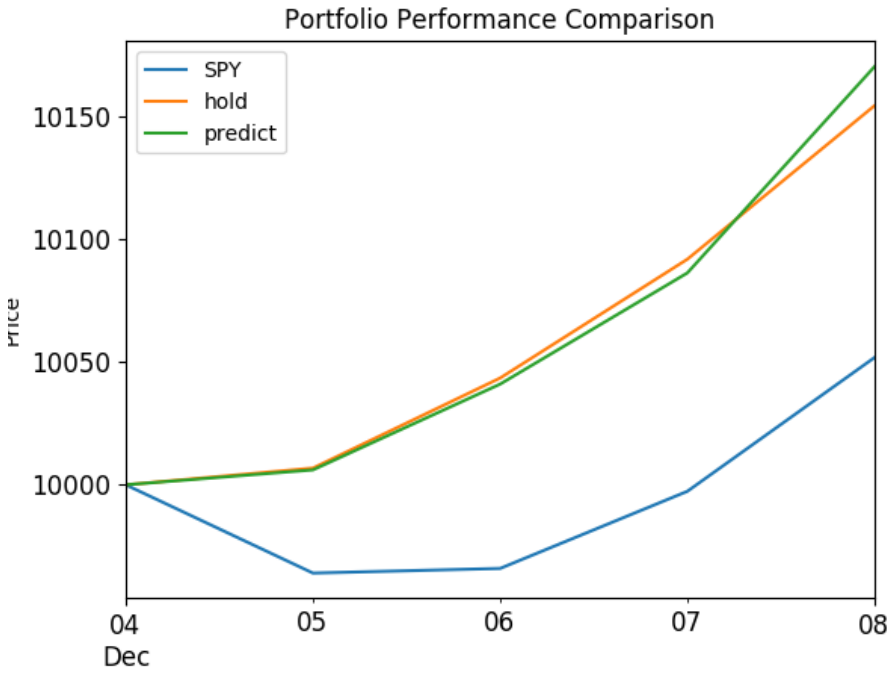
Features Used:

- Bollinger Bands®: two standard deviation away from the stock simple moving average
- Daily Returns

Result:

The following are the model results when used to actively trade the optimized portfolio between 2017-12-04 and 2017-12-08. In this new scenario, we can see that the active trading strategy barely beats the holding strategy of the same portfolio while both beat the S&P 500 index.

S&P	Hold strategy	Active trading Strategy
0.52%	1.35%	1.67%



Performance of three trading strategies compared

Reduce Training date:

Using training data between 2017-11-01 and 2017-12-01 and all of the features available, the profile optimizer generated the following set of symbols and their corresponding weights:

DE	NRG	HRL	M	SBAC	AES	BSX	KLAC	BBY	NCLH
23%	15%	15%	14%	11%	10%	8%	3%	1%	0%

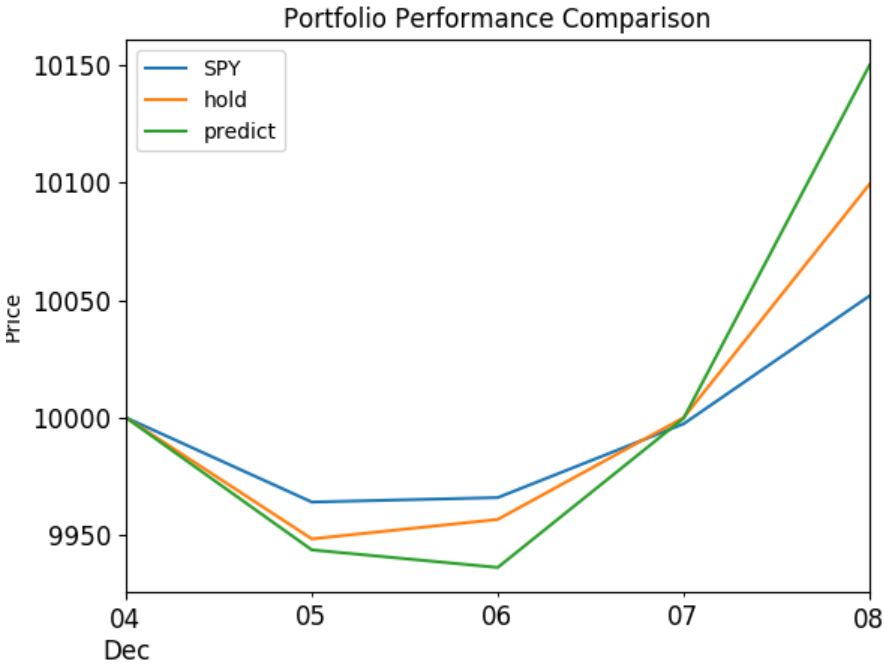
Features Used:

- Bollinger Bands®: two standard deviation away from the stock simple moving average
- Momentum: the rate of acceleration of a stock's price.
- 5 day Simple Moving Average
- Daily Returns

Result:

The following are the model results when used to actively trade the optimized portfolio between 2017-12-04 and 2017-12-08. Those results are also compared to both the performance of holding the portfolio during the same period and holding the S&P 500 stock index during the same period. The results clearly show that the active trading strategy is beating the other strategies in the span of 1 week with 1 month of training data, but fails to impress when compared to using an optimized portfolio using 11 months of data.

S&P	Hold strategy	Active trading Strategy
0.52%	0.96%	1.57%



Performance of three trading strategies compared

Reduced feature set with 1 month training data

Features Used:

- Bollinger Bands®: two standard deviation away from the stock simple moving average
- Daily Returns

Result:

Using 1 month of training data, reducing the feature set seems to have no impact on the results.

S&P	Hold strategy	Active trading Strategy
0.52%	0.96%	1.57%

11 month training data in 2015

Using training data between 2015-01-05 and 2015-11-06 and all of the features available, the profile optimizer generated the following set of symbols and their corresponding weights:

AMZN	GPN	ATVI	AYI	HOLX	NFLX	CI	MNST	NVDA
22%	21%	20%	10%	9%	6%	6%	3%	3%

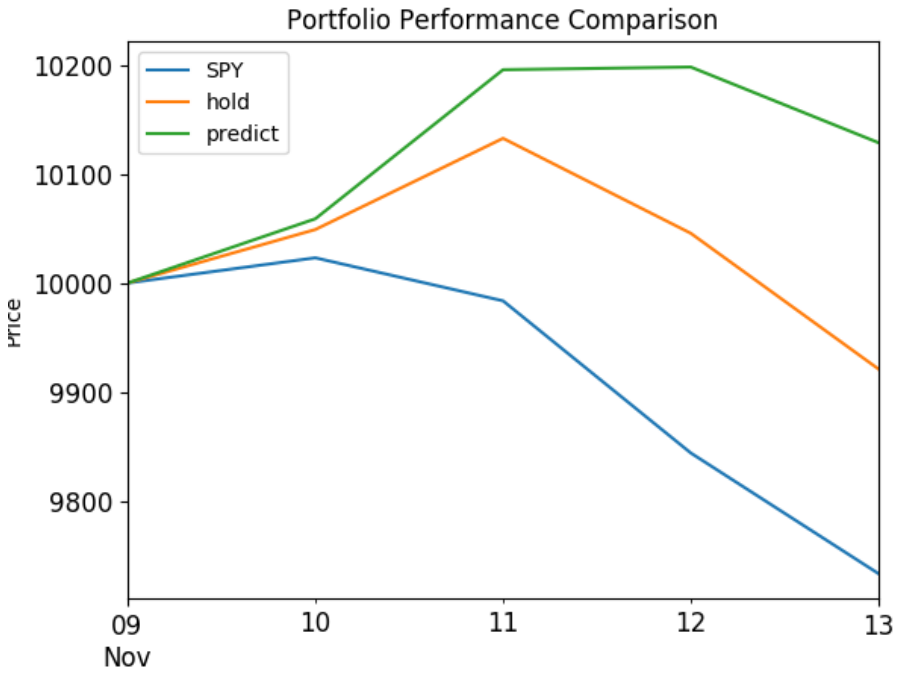
Features Used:

- Bollinger Bands®: two standard deviation away from the stock simple moving average
- Momentum: the rate of acceleration of a stock's price.
- 5 day Simple Moving Average
- Daily Returns

Result:

The following are the model results when used to actively trade the optimized portfolio between 2015-11-09 and 2015-11-13. The results clearly show that the active trading strategy is beating the other strategies in the span of 1 week with 11 months of training data.

S&P	Hold strategy	Active trading Strategy
-2.64%	-0.7%	1.31%



Reduced feature set with 11 month training data

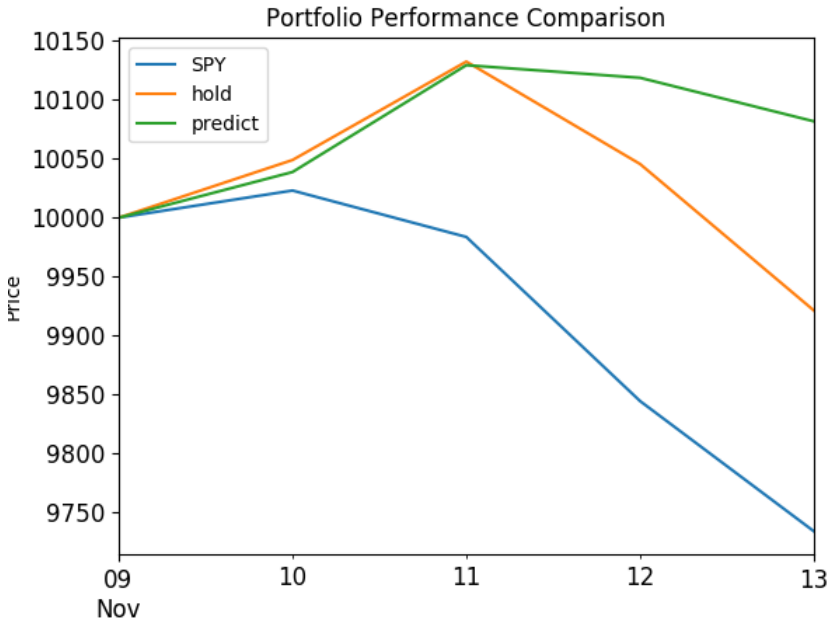
Features Used:

- Bollinger Bands®: two standard deviation away from the stock simple moving average
- Daily Returns

Result:

Using 11 month of training data, reducing the feature set seems to have a negative impact on the portfolio result between 2015-11-09 and 2015-11-13 while still beating the other two strategies.

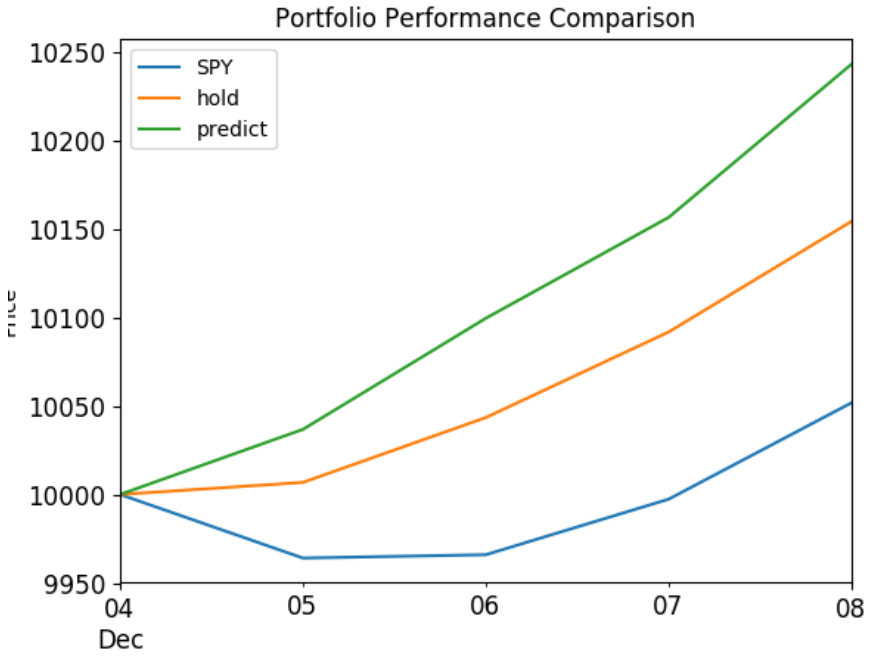
S&P	Hold strategy	Active trading Strategy
-2.64%	-0.7%	0.8%



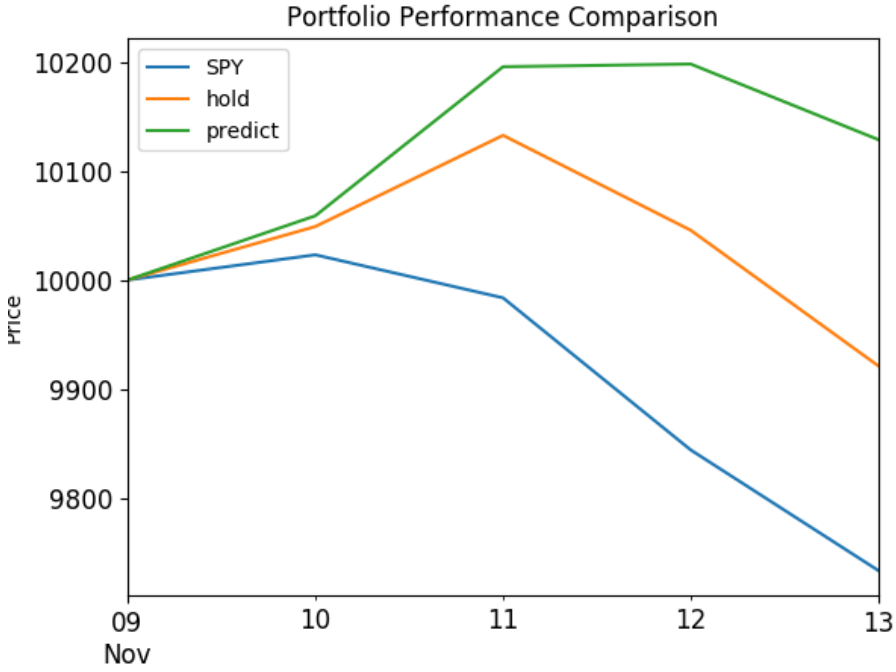
Justification

Based on the results so far, it is clear to me that using all the available features and 11 months or so of training data seem to generate a Linear Regression model that clearly outperforms my benchmark. Assuming that there are no trading fees, the active trading strategy has clearly outperformed the S&P 500 in the span of 5 days. It also outperformed holding the optimized portfolio for the same duration. This leads me to trust using this strategy over passive investing in an index fund tracking the performance of the S&P 500.

I have to say that I did not expect to come up with a model that outperforms the S&P500 on most days.



2017-12-04 to 2017-12-08.



2015-11-09 to 2015-11-13

V. Conclusion

Reflection

The question I asked myself at the beginning of this journey was if it is possible to create an application that can provide stock picks for a short-term investor without any human intervention. This application would have to be able to get up to date historical data on stock market, generate a balanced portfolio, then train itself with the most up to date information and generate a model that can be used to determine if it is time to buy the stocks that make up the portfolio.

There were many challenges facing me in getting an answer to this question. Some of them included asking myself which data can we use, which features should we try, how to generate those features, what algorithms can we use and which parameters can we change to get better results and finally how can we determine that this model does work? It took me weeks to get up to speed on what can be done in this area. This domain in my opinion is far more challenging than solving some of the other machine learning problems, but I was determined to learn more about it. This drove me to spend so many hours learning and coding the solution for this problem. I started with the Udacity course for machine learning in trading. I then collected whatever resources I can find online about solving similar problems in this space. My solution was largely inspired by the teaching of the course professor Tucker Balch and the work of some of his students.

The answer to that is yes, it can, if you are willing to take a risk. The results are mostly good but as we saw on the trials, my investment strategy does not beat the market every day, but it does most of the time.

Improvement

Predicting stock prices is an elusive and exciting topic. You will always find people who are trying to get a leg up in predicting and beating the market. I am confident that there is a way to truly predict a stock price, both short and long term. We are close but not close enough. The key would be to add more features to the model, such as sentiment data from social media, company book value, quarterly results and maybe even data from other markets both domestic and international. The challenge would be to properly collect and preprocess this data. I am also interested in the potential of using Neural Networks to solve the price prediction problem. I cannot wait to start experimenting with these different algorithms and techniques to solve this and other problems.

References

[1] Sentdex- Pythton for Finance https://www.youtube.com/watch?v=W4kqEvGI4Lg

[2] Arwarner- Machine learning for trading https://github.com/arwarner/machine-learning-for-trading

[3] Siraj- Predicting Stock Prices https://www.youtube.com/watch?v=SSu00IRRraY

[4] voyageth- Machine learning for trading
https://github.com/voyageth/trading/blob/master/capstone.ipynb