Predicting the Daily Direction of the S&P500

Capstone Project

Udacity Machine Learning Engineer Nanodegree

Pavel Liashkov

August 2019

I. Definition

Overview

On any given day, there is approximately a 50% chance that the stock market will increase and a 50% chance that the stock market will decrease. If a trader was able to predict which direction the market would move on any given day, with a probability significantly greater than 50%, that trader would have a significant competitive advantage and could use this insight to make better trading decisions.

There are several indices for monitoring the performance of the US stock market. In this project, the S&P500 index, which measures the performance of the stock prices of 500 of the largest publicly traded companies, will be used. Historical data for the S&P500 is widely available. For this project, over 26 years of historical S&P500 data was downloaded from Yahoo! Finance.

I choose this domain because I work in Forex broker from 2016 as SSE. I have many task in financial indicators. I have some experience in this domain.

Problem Statement

The objective of this project is to determine whether machine learning techniques can be used to predict the directionality (i.e. up or down) of the S&P500 stock market index on any given day, based on the market opening price and an assortment of historical market data. For this project, directionality will be defined as a ‘1’ for an increase in market value and as a ‘0’ for a decrease in market value, so the problem will be a machine learning ‘classification’ problem. The directionality is defined relative to the opening value. For example, if the market opens at 2,170 at 9:30ET and closes at 2,175 at 16:00ET, that would count as an increase and directionality value of ‘1’.

Several supervised learning classification techniques will be used to address the

objective, namely Random Forest, Gaussian NB, XGBoost, SVM and a neural network based

approach. The specific approach will be as follows: using market price, market volume and market volatility data, various data features will be calculated ('engineered'). These features will primarily focus on relative returns and relative differences between the trading days, across various data columns. Using these features, the machine learner will be trained and optimized to predict the label for any given day, where the label is a '1' if the market is predicted to close higher that day and a '0' otherwise.

Metrics

The stock market has historical increased slightly more than it has decreased (approximately 52-54% of day's show increases). Therefore, rather than randomly choosing up or down as the benchmark, with a 50% probability of either, a more suitable benchmark is always predicting an increase (i.e. always choose a directionality of '1'), since this will be correct 52-54% of the time, depending on the period under consideration.

The accuracy of the machine's predictions will be compared to the accuracy of the benchmark case. Accuracy is defined as follows:

$$Accuracy = (True\ Positives + True\ Negatives) / Total\ Population$$

In addition, the F1 Score of the benchmark and the model will be compared, since F1 provides the added insight of precision and recall. The F1 Score is defined as follows:

$$F1 = 2 \times (Precision \times Recall) / (Precision + Recall)$$

Where:

$$Precision = True\ Positives / (True\ Positives + False\ Positives)$$

$$Recall = True\ Positives / (True\ Positives + False\ Negatives$$

As a final metric for assessing the models, the machine learner will be used to make stock trading predictions, to determine whether any increase in accuracy versus the benchmark can be exploited for trading profit. Specifically, the daily profit from the benchmark and the machine learner will be tallied to generate a cumulative profit over the test period, allowing one to assess whether the recommendations of the model can be converted into a useful trading strategy. Since the ultimate objective of a stock trader is to make profitable trades, using

cumulative profit is a justified metric to convert accuracy results into trading performance.

II.Analysis

Data Exploration

The data used in this project was historical market data from August 16th, 1993 to September 14th, 2019. The start date was chosen based on this date being the earliest date for which volatility (VIX) data was available via Yahoo! Finance. The VIX is considered the market's 'fear index' and measures the volatility of stocks. During times of market turmoil, the VIX increases.

The raw data is included in the 'SP500_historical.csv' file provided as part of the project submission. The raw data file contained 10 fields, and as will be discussed later, these 10 fields were used to derive the features for the machine learner.

The 10 raw data fields are as follows:

- Date: calendar date for any given data row
- SP_Open: opening value (recorded at 9:30ET) for the S&P500
- SP_High: highest value on any given day for the S&P500
- SP_Low: lowest value on any given day for the S&P500
- SP_Close: closing value (recorded at 16:00ET) for the S&P500
- SP_Volume: number of shares of S&P500 components traded
- Vix_Open: opening value for the VIX index
- Vix_High: highest value on any given day for the VIX index
- Vix_Low: lowest value on any given day for the VIX index
- Vix_Close: closing value for the VIX index

For the date range under consideration, the raw data file contains 5,962 trading days. Figure 1 presents a plot of the S&P500 and VIX closing value for each of these days.

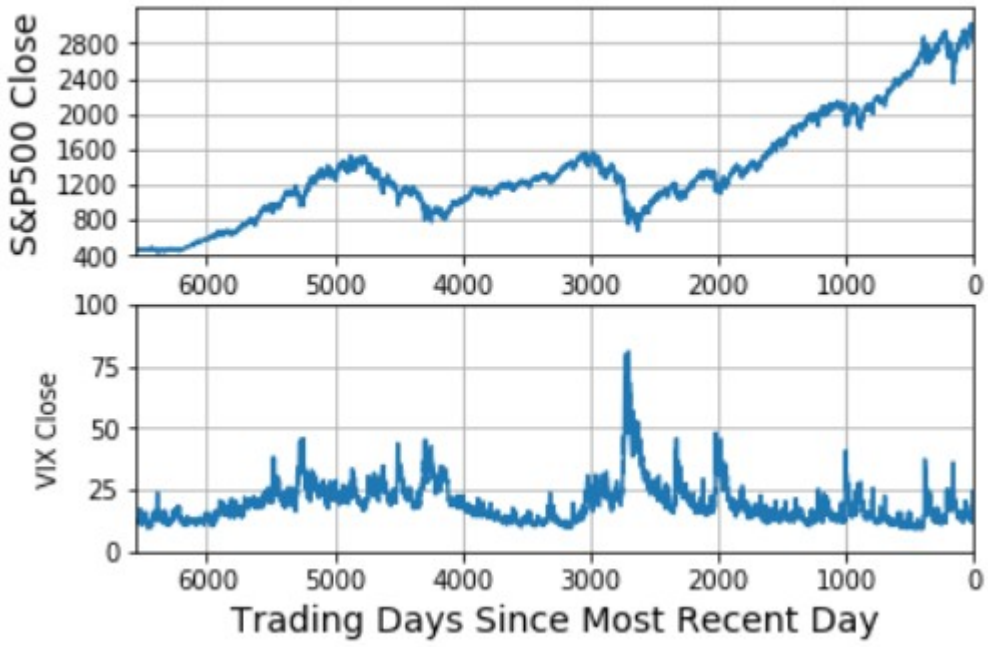


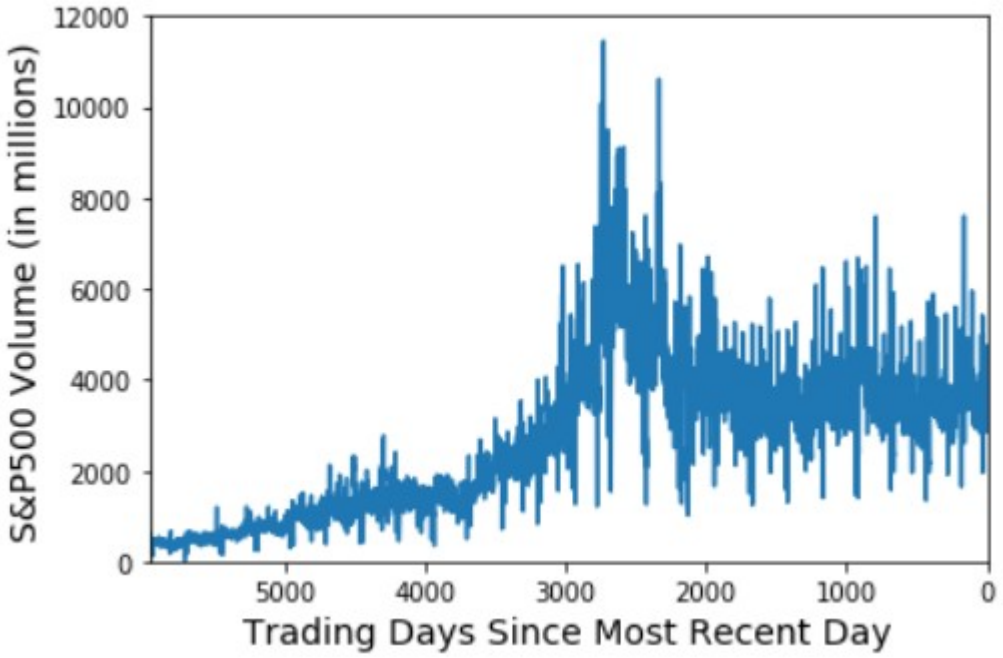
Figure 1: Data Exploratory Visualization: S&P500 and VIX Closing Values

The S&P500 has generated considerable up and down movement over the 26 years under consideration, but in general, has trended upwards. For example, the value at the start of the dataset was approximately 400 whereas at the end of the dataset, it had risen to approximately 2,200. Given this significant range in values, the data should and will be normalized prior to machine learning analyses, so that relative data can be used to identify relevant trends across such a wide time range.

In contrast, the VIX index does not generally trend upwards. At any given time, it indicates the level of volatility in the market, and thus can spike and then subside. For example, the massive spike in the VIX from around 2,000 trading days ago corresponds to the 'Great Recession' of 2008.

The trends seen for the other S&P500 and VIX related data (opening, high and low values for each) are very similar to those shown in Figure 1. The only other data in the raw data file is the S&P500 trading volume for any given day, and that data is presented in Figure 2.

Figure 2: Data Exploratory Visualization: S&P500 Trading Volumes



The S&P500 trading volume has generally trended up with time, although it peaked around the time of the 'Great Recession' and appears to have come down and stabilized since then.

Given the properties of the raw data, normalization steps will be used for transforming all of the data to percentage changes relative to other days, as will be discussed in detail in the Methodology portion of this report. This type of normalization is required given the huge differences in absolute values for each of these data sets over the 20+ years under consideration here. In addition to normalization, new features will be 'engineered', based on various calculations conducted on the raw data set, ultimately arriving at 20 features for the machine learning algorithms.

Algorithms and Techniques

Four supervised learning techniques were applied in this project:

1. Random Forest (RF)
2. Naïve Bayes (NB)
3. Support Vector Machines (SVM)
4. Xgboost (XGB)
5. Neural Network (NN)

The first three techniques are available in the Python scikit-learn (sklearn) module. The NN was built using TensorFlow. I will now discuss each of these four techniques in a little more detail and also why I was motivated to use them in this project.

Random forest, like its name implies, consists of a large number of individual decision trees that operate as an ensemble. Each individual tree in the random forest spits out a class prediction and the class with the most votes becomes our model's prediction. The fundamental concept behind random forest is a simple but powerful one — the wisdom of crowds.

Naive Bayes is a probabilistic based algorithm and has proven to be generally applicable to document classification and spam filtering problems. Strengths of NBs are that they require relatively small data sets and are extremely fast learners and classifiers. Like RFs, they are also relatively easy to understand and interpret, especially for people accustomed to thinking in terms of probabilities. Weaknesses include low accuracy in broader applications (i.e. beyond document classification) and low toleranace to highly interdependent attributes.

Support vector machines are generally applicable to learning problems with continuous attributes (not discrete ones). Strengths of SVM include relatively high accuracy and a high tolerance to irrelevant, redundant or interdependent attributes. Weaknesses of SVM include the requirement for large data sets, slow learning speed, risk of overfitting and models that lack intuitive understanding, making them hard to understand, interpret or explain to others (i.e. they are very "black box" in nature).

XGBoost is an implementation of gradient boosted decision trees designed for speed and performance. Boosting is a method of converting weak learners into strong learners. In boosting, each new tree is a fit on a modified version of the original data set. The Boosting Algorithm begins by training a decision tree in which each observation is assigned an equal weight. After evaluating the first tree, we increase the weights of those observations that are difficult to classify and lower the weights for those that are easy to classify. The second tree is therefore grown on this weighted data. Here, the idea is to improve upon the predictions of the first tree. Our new model is therefore Tree 1 + Tree 2. We then compute the

classification error from this new 2-tree ensemble model and grow a third tree to predict the revised residuals. We repeat this process for a specified number of iterations. Gradient boosting algorithm uses gradient descent method to optimize the loss function.

Neural networks are a powerful technique based on the workings of brain neurons. Their advantages include impressive results in many domains and also broad applicability. The disadvantages include their 'black box' nature, making it difficult to determine the impact of individual features, the large amount of data often needed, the large number of hyper parameters requiring tuning and also the significant run time often required to achieve good results.

I chose these approaches because they are quite different from each other. No single algorithm can uniformly outperform other algorithms over all datasets, so I felt that I would like to use four distinct methods to get a feel for which one would work best. I knew that the data set would ultimately include some discrete / categorical features, which would tend to favor RFs and XGBs. However, I also knew that I am dealing with a relatively small training set (2,000 - 5,000 samples), whichplays to one of NBs strengths.

Benchmark

Over the last year, which will serve as the test set for the models, the S&P500 closed higher on approximately 51% of trading days. Therefore, absent any other data, the most reasonable prediction for any given day would be that the market would increase (i.e. a label value of '1'). If the model were to make this prediction on the test set used, it would have an accuracy of 51% and an F1 score of 0.59 (calculations shown in the accompanying Python notebook). These values will serve as the benchmark for this project.

I. Methodology

Data Preprocessing

A significant amount of data processing and feature engineering was carried out on the raw data set, to transform the 10 data fields into one label and 20 features. There were several reasons for these transformations. First, as discussed previously using absolute values for the various data fields would likely not be productive, given how much the S&P500 index has grown (over 5X) in the 26 years worth of data used here. Therefore, fields needed to be turned into relative values (e.g. relative returns, relative volumes) to make it possible to compare data points on equal footing. Second, a fair amount of domain knowledge needed to be built into the models. I have spent the last nine years of my career as an options trader, and during that span, have developed insights into what I believe are important characteristics for predicting stock movements. I wanted to attempt to capture some of this domain knowledge in my features, which required further transformation.

The one label that was calculated for the project was called 'Intraday_Increase'. 'Intraday_Increase' takes on the value of '1' if the S&P500 closed higher than it opened on any given day and '0' if it closed at or below the opening value. This is determined by taking 'SP_Close' minus 'SP_Open' and assigning a '1' if the value is positive and a '0' otherwise. The objective of the machine-learning algorithm is to predict this label.

The 20 features that were calculated for this project were as follows:

1. Days_Since_Open
2. Break_Coming
3. Overnight_Return
4. Overnight_VIX
5. O_to_O

6. Trail_1d_Ret
7. Trail_2d_Ret
8. Trail_3d_Ret
9. Trail_4d_Ret
10. Trail_5d_Ret
11. Trail_21d_Ret
12. Trail_63d_Ret
13. Trail_126d_Ret
14. Trail_252d_Ret
15. Trail_1d_VIX
16. Trail_5d_VIX
17. Trail_1d_Rel_Vol
18. Trail_5d_Rel_Vol
19. Trail_1d_PtT
20. Trail_1d_VIX_PtT

Each feature will now be discussed in further detail.

'Days_Since_Open' is a measure of how many days it has been since the market was last open. On a regular week, Tuesday, Wednesday, Thursday and Friday will have a value of '1' whereas Monday will have a value of '3'. A day following an intraweek holiday should have a value of '2' and a Monday or Tuesday following a long weekend should have a value of '4'. In rare circumstances (e.g. following the September 11, 2001 terrorist attacks) the market could be closed for even longer and this duration will be reflected in 'Days_Since_Open'. 'Days_Since_Open' was calculated by subtracting the dates between two adjacent rows. The rationale for including this feature is that the markets might behave differently depending on whether traders have had a long break to digest news. Also, market moving information is routinely released over weekends (e.g. very large mergers, government bailouts etc.).

'Break_Coming' is a Boolean measure of whether or not a weekend or holiday is approaching. On a regular week, Monday – Thursday will have a value of '0' whereas Friday will have a value of '1'. On a holiday week (e.g. Thanksgiving Thursday), a regular weekday (Wednesday in this example) should have a value of '1'. 'Break_Coming' values were derived by shifting 'Days_Since_Open' by one row and looking for instances where 'Days_Since_Open' values exceeded '1', since these indicate weekends or holidays. The rationale for including this feature is that traders' moods and outlook are possibly dependent on pending breaks.

'Overnight_Return' is a measure of the percentage return when comparing 'SP_Open' of day n to 'SP_Close' of day n-1. It represents whether the market

has opened higher (if 'Overnight_Return' is positive) or lower than the previous close. Since the opening price (SP_Open) is the last and most recent piece of data that the machine will have prior to having to predict whether the market will move up or down from there, using it in various features is critical.

'Overnight_VIX' is analogous to 'Overnight_Return', except that it uses the values for the VIX Index instead of the S&P500 Index. An increase in the VIX overnight might be taken as indication that the market will be more volatile that day.

'O_to_O' is a measure of the percentage return when comparing 'SP_Open' of day n to 'SP_Open' of day n-1. It represents whether the market has opened higher (if 'O_to_O' is positive) or lower than the previous day's open. A positive 'O_to_O' value could indicate market optimism and resilience and provides yet another opportunity to incorporate the crucial 'SP_Open' value into a feature.

The next nine features, namely 'Trail_1d_Ret' to 'Trail_252d_Ret' all involve a near identical calculation, with the exception of the number of days back in time one looks. These are all trailing return indicators. For example, 'Trail_1d_Ret' measures the market return for trading day n-1 compared to trading day n-2, by calculating the percentage change in 'SP_Close' between the two days. This provides a very *short-term* indication of how the market has performed on the most recent historical day. Similar calculations were done looking back 2, 3, 4, 5, 21, 63, 126 and 252[2] trading days. By including a wide range of trailing returns, the machine-learning algorithm can gain insight into a wide range of recent market performance, from the very short-term to the fairly long-term.

'Trail_1d_VIX' and 'Trail_5d_VIX', are derived by using 'Vix_Close' values and performing the same type of trailing return calculations as described above for the S&P500 index. These two features measure whether market volatility is trending up or down in the very short-term and short-term.

'Trail_1d_Rel_Vol' is a measure of whether the most recently available volume data (i.e. the data from day n-1) is above or below average compared to the previous four days before it. It takes the day n-1 volume, divides it by the rolling average of the 4 days prior (days n-2 through n-5), and then subtracts 1 from the result. Thus, if the value is above zero, the most recent day has relatively high volume compared to the prior 4 days. The rationale for including this feature is that it seems important for the machine learning algorithm to know whether trading volume on the most recent day was relatively heavy, relatively unchanged or relatively light compared to the previous four days.

[2] Once weekends and holidays are factored in, there are approximately 21 trading days in most months and approximately 252 trading days in most years, and hence the 252 day trailing return represents the most recent one-year performance.

‘Trail_5d_Rel_Vol’ is a very similar measure to ‘Trail_1d_Rel_Vol’ but it looks at volume data over a longer period. It takes the rolling average of the previous 5 days of volume data and compares it to the rolling average of the 21 days prior to those 5 days. In other words, it is effectively determining whether the most recent week of trading volume is heavy compared to the prior month (since a typical trading week is 5 days and a typical month is 21 days). This feature could be an important indicator of whether the market is experiencing a busy or quiet period, and hence a useful feature in predicting movements.

‘Trail_1d_PtT’ is the only feature that makes use of the ‘SP_High’ and ‘SP_Low’ data in the raw data file. ‘PtT’ stands for peak to trough. This feature is an indicator of how much the previous trading day moved from its peak (‘P’) to its trough (‘T’). To derive this value, one takes the previous day’s high value (‘SP_High’) and subtracts the previous day’s low value (‘SP_Low’) and then divides the result the average opening and closing prices (‘SP_Open’ and ‘SP_Close’). On days where the difference between peak and trough are small, this feature will approach zero, but on days with significant differences between the peak and trough, this value could be well above zero. The rationale for including this feature is that it allows the machine learner to know whether the previous trading day experienced significant swings in market value or not.

The final engineered feature, ‘Trail_1d_VIX_PtT’ performs an identical calculation to that discussed for ‘Trail_1d_PtT’ except that VIX raw data is used instead of S&P500 data.

In addition to the data transformation and feature engineering steps discussed above, several rows of data were discarded after completing these steps. First, any row with a ‘nan’ value was discarded. These ‘nan’ values were an expected outcome of the feature-engineering step. Specifically, the 252-day trailing return calculation could not be properly done on the first 253 data points, since there were no historical data on which to base the calculations. **Applying ‘.dropna’ to the 6547 row data set resulted in 253 drops and hence a data set consisting of 6294 rows.**

Second, it was observed that for a large number of the older data points (those points past approximately 8 years ago), the ‘SP_Open’ value on day n was identical to the ‘SP_Close’ value on day n-1. Since the probability of the S&P500 opening on any given day at a value **exactly** equal to the former day’s close is essentially zero, this data was considered unreliable. It is possible that the previous close was simply being pasted in as the new open for a large chunk of the data, due to a data generation anomaly. Therefore, any data row where the SP_Open was exactly equal to the previous day’s SP_Close field was discarded.

This was achieved by creating a new column called ‘No_Overnight_Change’, which was set equal to ‘1’ if the opening value is identical to the previous day’s closing value, allowing one to drop all of those rows from the final data set fed to the machine learners. The final data set consisted of 3330 rows.

Implementation

Four supervised machine-learning methods were used in this project:

1. Random Forest Classifier from sklearn
2. Gaussian Naïve Bayes from sklearn
3. Support Vector Machine SVC from sklearn
4. XGBoost
5. Neural Network using TensorFlow

In all four cases, the data was split as follows. The most recent 252 days (approximately one year) was used as the test set. The balance (3078 rows) was used as the training set. In the case of the neural network, the training set was further split into a training set (2574 rows) and validation set (504 rows). It should be emphasized that the data was NOT shuffled prior to splitting between the test, validation and training sets. The reason for doing so is that the data set is a time series, so the most recent days are segregated for testing purposes, to avoid any risk of letting the machine learners see 'the future', which would be a risk if the data set was shuffled up and the test set was derived from some data that occurred earlier in time than the training set.

The three sklearn algorithms were initially implemented using the default settings. They were tested against different training set sizes, to gauge the impact of data set size on the various model's performances. Later, grid search cross validation

(GridSearchCV via sklearn) was conducted on the RF, XGB and SVM models, in order to optimize the hyper parameters.

The neural network model was loosely based on the TensorFlow based MNIST assignment. Prior to running the model, all features were scaled (mean = 0; standard deviation = 1) and 1-hot encoding was conducted on the label data. The neural network consisted of 3 layers, containing 10, 5 and 2 nodes respectively. The layers weights were initialized with a mean of zero and standard deviation of 1. L2 regularization was added, with a lambda value of 10^{-5}. Learning rate used exponential decay, with a starting learning rate of 0.05, a decay_steps setting of 100,000 and a decay_rate of 0.98. Dropout was incorporated, with a default drop probability of 0.5 for each of the three layers. The default number of steps used was 10,000. Later, these various hyper parameters were tuned to improve the model's performance.

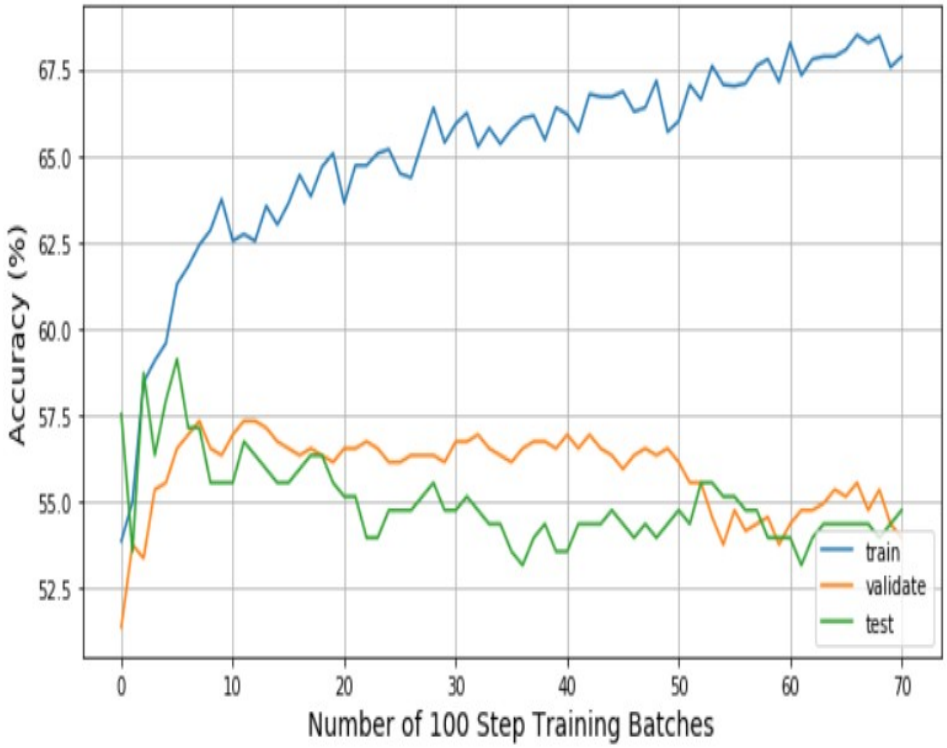


Figure 3: NN, Train, Validation and Test Accuracy versus Training Steps

Refinement

The initial results for the four learning algorithms are presented in Table 1.

Table 1: Initial Learning Performance Results

Model	Accuracy (%)	F1 Score
Benchmark	53.57	0.6977
Random Forest (RF)	53.57	0.5551
GaussianNB	55.16	0.6367
XGBoost	54.76	0.6149
SVM	53.57	0.6977

Benchmark F1 score results from simply predicting all '1' on the test set.

SVM default result is equal benchmark, it is not error, it is trand.

Most of the models exhibited accuracy values at or barely above the benchmark and F1 scores at or below the benchmark. The only promising result was the SVM accuracy.

To improve on these results, grid seaclean_dfrch cross validation was performed on the RF, XGB and SVM models.

For the RF model, it was discovered that the best accuracy 56,35% accuracy, with an F1 Score of 0.6474.

I was tried these parameters:

- n_estimators = range(5, 15, 1)
- max_depth = range(1, 12, 1)

Best accuracy was achieved with parameters max_depth = 4, n_estimators = 14.

For SVM model - 56,35% accuracy, with an F1 Score of 0.6726.

I was tried these parameters:

- C=[1, 10, 15, 20]
- gamma = [0.0001, 0.00001, 0.00001]

Best accuracy was achieved with parameters C=13, gamma = 0.0001.

For XGB model – 55,95% accuracy, with an F1 Score of 0.6159.

I was tried these parameters:

- max_depth = range(1, 12, 1)
- n_estimators = range(50, 600, 50)

Best accuracy was achieved with parameters max_depth = 2, n_estimators = 250.

For choosing other XGB parameters I use this method for choose best parameters https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/

The results from the tuned models are compared to the benchmark in Table 2.

Table 2: Tuned Learning Performance Results

Model	Accuracy (%)	F1 Score
Benchmark	53.57	0.6977
Random Forest (RF)	56.35	0.6474
GaussianNB	55.16	0.6367
SVM	56.35	0.6726
XGBoost	55,95	0.6159
Neural Network (NN)	55.56	0.5851
VotingClassifier (RF + SVC + GNB)	55.95	0.6520

IV. Results

The purpose of the machine-learning model developed here is to predict whether the S&P500 will close higher than its opening price, on any given day. Ultimately, the goal would be to make profitable trading decisions using the model's insights.

By simply always guessing that the market will close higher, one would be correct 53.57% of the time for test set, whereas the NN based model would allow one to be correct 55.56% of the time. The question is whether this 2% improvement in accuracy can be turned into a profitable trading strategy. To address this question, a simulated trading scenario was conducted. If the model predicted a day would close lower, then the S&P500 index was sold at the opening price at the open of the market and bought at the closing price at the close of the market (i.e. short sell), leaving the trader with the cash difference between the two prices. The reverse strategy would be used if the model predicted a day would close higher. This strategy was then repeated everyday for the 252 day test set. Two different benchmark scenarios were used. In one benchmark scenario, the trader always bought the index at the open and then sold the index at the close. In the other benchmark scenario, the trader bought the index on day 1 and then held the index until the end of the test period (day 252), finally selling the index at the closing price on the last day[7].

The trading scenario was simulated and the resulting cumulative profit is presented in Figure 4. The difference between the benchmark strategies is stark. By the end of the 252-day test period, the neural network trading strategy has a cumulative profit of $1086.74. The strengths of the neural network predictions do not become apparent until after the first month or so (approximately look back day 230 on the chart), when the market starts to decline and the buy only based strategy stagnates or declines. At this point, there is a big reward for being able to accurately guess which days will see declines in value, and selling at the open, which is often what the neural network model is able to achieve. Performance statistics for the daily
profit of the strategies[8] are summarized in Table 3.

[7] For completeness, it is worth noting that two more benchmarks could be included, wherein, the trader always sells each day at the open or sells on day 1 and holds until the last day. The results would be mirror images of the existing two benchmarks, so were excluded here.

[8] The 'buy and hold' strategy is not included, since the results are nearly identical to the 'buy only' strategy

Figure 4: Cumulative Trading Profit: Neural Network Model vs. Benchmarks

Table 3: Trading Strategy Daily Profit Statistics (all values in $)

Parameter	Buy Only	Neural Network
Mean	0.06	4.32
Standard Deviation	22.52	22.12
Minimum	-88.22	-47.71
25^{th} Percentile	-8.10	-7.10
Median	0.91	2.26
75^{th} Percentile	11.27	12.40
Maximum	104.57	104.58

V. Conclusion

In this project, a machine learning neural network has been developed that predicts whether the S&P500 will close higher or lower than the opening price

Reflection

Attempting to predict the daily movement of the stock market can be reduced to a regression problem (e.g. trying to predict the actual price at which a stock or index will close) or a classification problem (e.g. trying to predict whether the market will close higher or lower, without too much concern for the actual final value). This project has attempted to tackle the problem in the classification form, as this is a question that has long been of interest to me in my professional career as an equity options trader, where directionality is often of more interest than the actual magnitude of the movement.

In order to address this question, a collection of powerful machine learning classification methods has been utilized here. This collection included three 'off the shelf' classifiers available via the Scikit-learn (sklearn) machine learning library, namely RandomForestClassifier, GaussianNB and svm.SVC. With these models, GridSearchCV was utilized to perform a search over the machine learning parameter space, which led to optimal accuracy results. These results

exceeded the benchmark accuracy by a meaningful amount and could be used to power successful trading strategies.

The effort was taken a step further, by building a neural network 'deep learning' model using the TensorFlow machine learning library. The model was manually tuned and ultimately delivered the best accuracy and F1 Scores relative to the benchmark case. This model was stress tested by evaluating its performance when trained (and not tuned) on a much larger set of market data. Under these conditions, it still performed well above benchmark and could power a trading strategy that delivered far superior profitability compared to a buy only strategy.

One particularly interesting aspect of the results was that relatively modest gains in accuracy (53% benchmark vs. 55% fully tuned neural network) could lead to dramatic gains in cumulative trading profitability over the course of the one-year test set. This is the result of the machine learning models not only avoiding, but capitalizing on, some of the market down days.

Tuning the neural network was a relatively difficult process compared to the other three classifiers. The built in GridSearchCV allowed one to fairly comprehensively test the performance of these classifiers, whereas the neural network optimization process was far more manual. It is likely that the final accuracy could be improved upon further, with further tuning and expert insight.

Improvement

I started the Introduction to Programming Nanodegree offered by Udacity and am now deep into the Machine Learning Nanodegree program. As I've worked on this capstone and scoured the Internet for answers to my numerous questions about building machine learning models, I am struck by the expertise of many of the practitioners in this field. Therefore, I have no doubt that my work here could be greatly improved upon by the right group of experts.

The main area I would like to see improvement on and an expansion in my own personal knowledge is a more systematic approach to optimizing the neural network. The tuning process that I used is incredibly manual and I would be amazed if there aren't better ways to tackle the problem and greatly expand the

search space. I barely began to tackle the question of network depth, choosing to use two hidden layers more or less arbitrarily.

The other area for improvement would be the feature-engineering portion of the project. There are so many other factors that could be important to market movement (e.g. even the weather might be a factor) and investing resources in this area could have significant payoffs in the models' performance.