Predicting Trends in the Stock Market

CSC 420 – Machine Learning

Jordan Turley

**Overview**

Predicting trends in the stock market is one of the most enticing, as well as most difficult problems in AI and machine learning. Today, over 75% of trades on the New York Stock Exchange are executed by a computer and determined by some algorithm. However, this is still an extremely difficult problem. If this were an easy problem, then everyone would be making millions on the stock market. To be able to predict a trend in the stock market would allow a person to make a trade faster than any human can and make money with little risk. While some analysts believe that it is impossible to predict the future of the stock market, I believe the stock market can be predicted to some degree. In this project, I attempted two different methods for stock prediction. The first method used three different models: logistic regression, k-nearest neighbors, and support vector machines. I attempted to predict the stock’s performance (increase or decrease) for the next day based on the closing prices from the past year. The second approach converted the data to a time series and used a recurrent neural network to predict a stock’s increase on a given day based on the closing price for a few days back.

**Big Idea**

My main goal of this project was to develop a script that can run relatively quickly on a regular computer to predict a stock’s trend for the next day, based on its previous prices. This would be a program that would be trained on the data from some specific day, then in the future, one day after the market closes, you could choose a few stocks that you are interested in, run them through the model, and it would tell you if the stock is predicted to increase or decrease tomorrow; i.e. if you should buy the stock in after-hours trading, or first thing tomorrow morning, and hold until the end of the day, or not bother to buy, as it will decrease in value tomorrow.

**Dataset**

The central part of this project was the dataset. I was able to download historical data for the stock market easily from Yahoo Finance. An example can be seen for Google here:

<https://finance.yahoo.com/quote/GOOG/history?ltr=1>

It was easy to select the range of dates I wanted and download a CSV file of the data. The CSV file contained the date, open, high, low, close, adjusted close, and volume for the day. I attempted to write a script to download several stocks at a time, but this did not work. I had to manually download the stock history that I wanted. I decided to use all the stocks that are included in the S&P 500. At first, I thought about using the stocks included in the Dow Jones, but this was only thirty stocks, and I wanted more data than this. I had to manually download each stock’s history for all 500 stocks. I downloaded the data on 4/20/18, and used the past year, so the prices ranged from 4/20/17 to 4/20/18.

There were five stocks that I left out of the dataset: BHF, BKNG, CBRE, HBAN, and WELL. The data for CBRE did not go back an entire year, and the data for the other four contained several values that were NULL. I didn’t have a good idea of what to do with these. Since it was only five that there were issues with out of 500, I just chose to exclude these.

The part I was most interested in using was the closing price. I wanted to predict if the closing price tomorrow will be higher or lower than it was today, based on the past year’s data. I wrote a small script to fetch only the closing prices out of the CSV file from Yahoo, as well as determine if the price increased or decreased from 4/19/18 to 4/20/18. Of course, I did not include the price on 4/20/18 in the prices, as this would completely give away the results.

I split the data into training and test sets. Because the closing prices range from only a few dollars for some stocks to thousands of dollars for others, it was very important to rescale the data. I used sklearn’s StandardScaler class to rescale the data. This made each stock have much more of its own weight rather than only the very expensive stocks having all the weight.

The dataset was very skewed: from 4/19/18 to 4/20/18, there were only 93 stocks that increased in price, while 407 decreased. In other words, the market went down this day. It may have been interesting to investigate this further, but I did not; it is possible that there were interest rate changes or some other governmental announcement, or that the market went down for no other reason than chance. This caused for interesting results, which I will analyze later.

**Logistic Regression, k-Nearest Neighbors, and Support Vector Machines**

The first three methods I attempted were logistic regression, k-nearest neighbors, and support vector machines. I wanted to see if a relatively simple, transparent model like these could do well at predicting trends in the market.

My dataset was modeled with each feature being the price of the stock as a specific day, and the class I tried to predict was simply increase or decrease on the next day. For example, X1 = price on 4/20/17, X2 = price on 4/21/17, … Xk = price on 4/19/18, and my class was increase or decrease from 4/19/18 to 4/20/18.

There was one major problem with using a model like this: these models assume that the features are independent. We can clearly see that these features will not be independent. If you know the price on one day, you can reasonably guess that the price on the next day or even several days after that will be reasonably close to the price on the first day. However, even with this contradiction, I still wanted to see how these models did on the problem.

I used grid search with each model to search over several parameters to see which performed the best. For logistic regression, I explored values for the penalty norm of L1 and L2, and values for C ranging from 0.1 to 2.0. For k-nearest neighbors, I explored values of k from 1 to 10, and distance metrics of Euclidian, L1, Manhattan, and Chebyshev. Finally, for support vector machines, I explored linear, sigmoid, rbf, and polynomial kernel functions, and values of C ranging from 0.1 to 2.0.

**Results**

To evaluate these models, and decide which performs the best, I used repeated k-folds with the training set. The results can be seen below:

|  |  |
| --- | --- |
| **Model** | **Success Rate (Score)** |
| Logistic Regression | 0.816 |
| k-Nearest Neighbors | 0.8208 |
| Support Vector Machines | 0.816 |

As we can see, k-nearest neighbors narrowly beats logistic regression and support vector machines. However, all three scores were high, and close enough together that any of the three could be used.

I evaluated k-nearest neighbors on the test set. The results can be seen below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **k-Nearest Neighbors** | **Precision** | **Recall** | **F1-Score** | **Support** |
| Decrease | 0.83 | 0.99 | 0.9 | 306 |
| Increase | 0.73 | 0.12 | 0.2 | 69 |
| Average | 0.81 | 0.83 | 0.77 | 375 |

|  |  |  |
| --- | --- | --- |
|  | **Predicted: Decrease** | **Predicted: Increase** |
| **Actual: Decrease** | 99 | 2 |
| **Actual: Increase** | 24 | 0 |

**Analysis**

If we only look at our success rate, it seems that our model is doing very well at classifying stocks. However, when we look at our other metrics, and especially our confusion matrix, we can see what is happening under the hood. Other than two instances, the model always predicted that the stock will go down, because this is what most of the other stocks did. We see that our model does a very good predicting that stocks will go down, but this is because most stocks went down on the day that I chose to collect data for. I think it was simply unlucky that zero of the stocks that were predicted to increase went up that day; if the randomness had been a bit different, we probably would have seen at least a few correct predictions of increase.

It is worth noting that on the training data, k-nearest neighbors did predict some stocks to increase when they did; however, it was very few. If you followed this model on the training data, you would have bought eleven stocks and eight of them would have gone up. It is also worth noting that I did look at the metrics for logistic regression, and it was even worse; logistic regression always predicted the stock to decrease the next day, for both the training and test sets. Since 80% of the stocks went down that day, predicting a stock will decrease was correct 80% of the time. However, this is less than satisfactory, and we would like to see results that are more varied. We explore these in the next section.

**Recurrent Neural Network**

The first part of my model used a naïve and ultimately false assumption that each feature was independent. The second part of my project used recurrent neural networks to represent this data as a time-series and attempt to predict a stock’s price based on its last few closing prices. This model acknowledges that these prices are directly linked and rely on each other and tries to predict an increase or decrease based on this.

Recurrent neural networks are much like classical neural networks with one major change. Classical neural networks, or feedforward neural networks, always feed the data forward through the network, producing a result that is independent from all other results. Recurrent neural networks allow for loops back through the network, allowing for what is basically a memory of the previous result.

To utilize memory, I used a type of recurrent neural network called a long short-term memory network. This type of network can ‘remember’ the result from a previous classification for a certain number of future classifications, using this to aid in prediction of a future result. This is how our time series data works: we can predict an increase or decrease for a certain day, but based on this day, we are able to make future predictions. The knowledge of a past prediction can aid in future predictions.

Rather than look at all the stocks, this problem made more sense to use a separate network for each stock. For each stock, I split the data independently into features and a class. The features were a certain number of closing prices for the past few days, and the class was increase or decrease for the next day. Fundamentally, this problem differs from the last. Rather than each stock being its own instance, each stock has several instances, as one instance only contains a few prices rather than the entire year of prices.

For each stock, I independently read in the past year of data from a file. I split the data into a training set, validation set, and test set. I built the neural network and trained the model on the training set. Then, I made predictions for the validation set to be used later for determining the best values of the hyperparameters. Finally, once the best hyperparameters had been determined, I made predictions on the test set.

There were several hyperparameters I wanted to find values of: the number of days to include in each instance, i.e. the number of days to look back when trying to determine an increase or decrease on the next day, the number of hidden layers, and the number of ‘blocks’ to include in each hidden layer. To determine these values, I used a grid search over several hyperparameter values. In the interest of time, I selected five stocks at random: AET, CHD, DUK, MSFT, and SCG. I tried looking back 1, 2, 3, 4, and 5 days to predict the next day. I tried blocks of size 1, 2, 3, 4, 5, and tried 1, 2, 3, 4, and 5 hidden layers. I looked at the average performance over the five stocks on the validation set, and decided on the hyperparameters based on the best performance.

Even with only five stocks this still took a long time. Five neural networks had to be trained and evaluated for each combination of look back values, block sizes, and hidden layer sizes. This resulted in 625 neural networks being trained overall. I trained each for 100 epochs.

After looking at all possible combinations, I found the following as the best values of hyperparameters:

|  |  |
| --- | --- |
| **Hyperparameter** | **Value** |
| Days to look back | 1 |
| Blocks | 5 |
| Hidden layer size | 3 |

**Results**

After finding these values for the hyperparameters, again in the interest of time, I randomly select 50 stocks from the S&P 500 to evaluate on their respective test sets. The results varied slightly, but I was most interested in the best performing and worst performing stocks. These can be seen in the following table:

|  |  |  |  |
| --- | --- | --- | --- |
| **Best Performing Stock** | **Success Rate** | **Worst Performing Stock** | **Success Rate** |
| MAT | 0.619 | LB | 0.381 |

Again, there is much more than just the success rate, so we will look at several other metrics as well as confusion matrices.

The results for MAT can be seen below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **MAT** | **Precision** | **Recall** | **F1-Score** | **Support** |
| Decrease | 0.62 | 1.00 | 0.76 | 39 |
| Increase | 0.00 | 0.00 | 0.00 | 24 |
| Average | 0.38 | 0.62 | 0.47 | 63 |

|  |  |  |
| --- | --- | --- |
| **MAT** | **Predicted: Decrease** | **Predicted: Increase** |
| **Actual: Decrease** | 39 | 0 |
| **Actual: Increase** | 24 | 0 |

As well as the results for LB:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **LB** | **Precision** | **Recall** | **F1-Scoore** | **Support** |
| Decrease | 0.0 | 0.00 | 0.00 | 39 |
| Increase | 0.38 | 1.00 | 0.55 | 24 |
| Average | 0.15 | 0.38 | 0.21 | 63 |

|  |  |  |
| --- | --- | --- |
| **LB** | **Predicted: Decrease** | **Predicted: Increase** |
| **Actual: Decrease** | 0 | 39 |
| **Actual: Increase** | 0 | 24 |

**Analysis**

Initially, after only looking at the success rate, it looks like the model performs decently for MAT, but not very well for LB. However, after looking at these metrics and the confusion matrix, we see that the models are not doing well at all. Unfortunately, in both cases, and presumably every case, we see that the model defaults to predicting one or the other for every case. In cases like MAT, it works okay in terms of success rate, but in another case like LB, it doesn’t work well.

There are several possible reasons for this to occur, but I don’t have a good answer why it did. The simplest is that only predicting one class for every instance results in a minimum for the loss function. It is likely that the loss function gets stuck in a local minimum when there is some global minimum that would optimally predict an increase or decrease on the next day. It is also possible that it needed to train for much longer before it would produce results that were interesting.

**Future Work**

As would be expected by my results, there is much that I would like to do in the future to attempt to improve these results.

* Explore a larger dataset; it would be interesting to look at more stocks, to go back farther in time, and to look at different time periods, rather than all at the same time.
* Because most of the stocks went down the day I chose to download my dataset, it would be interesting to look at a different time of the market, when there is a more even spread of stocks that increased and decreased, rather than most increasing or most decreasing. The day that I chose to look at was unlucky in having most stocks decrease.
* Explore other approaches to logistic regression, k-nearest neighbors, and support vector machines. Rather than using only the price history, we could use a more technical analysis approach, using features like the market capitalization, price-to-earnings ratio, dividend yield, moving averages of closing prices, and any other features of stocks we think could be useful.
* Try other models, such as naïve Bayes, a single perceptron, or a classical feedforward neural network.
* Try other hyperparameters for our recurrent neural network. We could try other architectures as well as different numbers of days to look back to try to predict the next day.
* Confirm that my implementation and usage of the recurrent neural network in the context of this problem is correct.
* Train the recurrent neural network for more epochs. It is possible that 100 epochs isn’t long enough to train for, so the neural network is not able to provide results that are interesting.
* Explore other time series models, like Hidden Markov Models. HMMs were something I wanted to explore but did not have enough time to try. My visible states would be increase or decrease on the given day, and the hidden state would be something like the true value of the stock; overpriced or underpriced. We could attempt to learn the transition matrices and predict if a stock is overpriced or underpriced from the series of increases and decreases we have seen.
* Attempt to predict a stock increase or decrease over a different amount of time: within the same day over a matter of hours or minutes, or over several days or weeks.
* Attempt to predict the actual price of the stock rather than simply an increase or decrease. This could be done using regression or recurrent neural networks, and the class would be a number rather than increase or decrease.

**Conclusion**

Overall, I am disappointed in my results. I was hopeful that I would be able to predict the stock market with at least some accuracy. However, there were several things that went wrong. I chose a bad day to collect my data on, as most of the stock market went down that day. Because of this, when using models like logistic regression, k-nearest neighbors, and support vector machines, they found it most efficient to predict that the stock will always go down; 80% of the time, this will be true. The results would have been significantly different if I had collected data on a different day where there was more of a balance between increases and decreases in stocks.

I also don’t think that my neural network was working the right way. It is unusual that it predicted always increase or always decrease even when this does not provide optimal results. I wonder if there is something incorrect in my implementation, or if more training was needed, or if there is some other issue influencing my results.

Even though my results were not what I expected or wanted, that does not mean that this project is useless. This shows much of what is so hard to predict about the stock market. On any day, the market can go up or down for any reason. Government announcements, natural disasters, or pure speculation or fear can cause the market to increase or decrease in a seemingly random way. To predict the stock market based on only prices is a very difficult task that no one has truly figured out yet. There is more to a stock than just numbers, and accurately predicting the market with little or no risk is still something that is highly sought after, but extremely hard if not impossible to do.

**Personal Note**

This is where I will talk about the project in a less formal way. When I began the project I was very hopeful that I would be able to predict some sort of trends in the market with some accuracy. I begin trying to predict if the stock will increase or decrease over the next month. However, I transitioned to the next day, as this would conceptually make more sense when dealing with a recurrent neural network. It took some time and research to first find a library, then to find a tutorial that gives a good explanation of something at least somewhat like what I am trying to do. I finally found one, which I will list below as my reference. I attempted to follow it but predict an increase or decrease rather than a number. I was disappointed in the results I found for the first three models but was more hopeful for recurrent neural networks, as the stock market lends itself to a time series solution. However, as we saw, they did not perform any better. I would have liked to investigate this problem further; I wonder if something is inherently wrong with my implementation and use of the model, but I did not have enough time in the end to investigate the problem further.

I wanted to implement a Hidden Markov Model, but ran out of time. I think that this would have been interesting to look at and try to use.

If I could change what I did for this project, I would have started earlier. I also would have tried to gain a better understanding of recurrent neural networks and implemented a hidden Markov model for this problem.

I am worried and hope that my poor results do not negatively impact my grade on this project. Even though the results were bad, I feel like I have learned a lot throughout this project.

After my work with Dr. Lamar over the summer, I had a general understanding of machine learning, but I wanted to learn more. I am so glad that I took this class, Statistical Modeling, Probability Theory, and Mathematical Statistics, so much that statistics and machine learning interest me enough to focus on machine learning if I go to graduate school, which I probably will. This study of data, patterns, and trends interests me, and I hope to continue this path.

<https://machinelearningmastery.com/time-series-prediction-lstm-recurrent-neural-networks-python-keras/>