Capstone Project - Stock Market Prediction with Machine Learning

SungGeun Kim
May 7, 2017

1 DEFINITION

1.1 PROJECT OVERVIEW

One of the popular questions that anyone who has interest in the stock market is "is it possible to predict the stock price?". There have been skeptical arguments against the predictability of stock prices [1]. The efficient market hypothesis (EMH) [2] states that it is impossible to "beat the market" because all the relevant information is already reflected in the price of stocks. Some people believe that the stock market is a chaotic system where a small difference in the initial condition results in widely different results [3]. This means that the stock price is predictable in a short term, but not in a long term. Others believe that the stock market shows a random walk behavior where the price of stocks are completely unpredictable [4]. However, these theories have been disputed [5, 6, 7] and people have shown some successes in predicting stock prices with machine learning algorithms such as support vector machines (SVM) [8, 9] or neural network [10, 11].

This project is an attempt to show that it is possible to use machine learning algorithms to predict the stock market directions at least better than the buy-and-hold strategy. For simplicity, the Standard & Poor (S&P) 500 stock index is used. The S&P 500 is an American stock market index that tracks the stock prices of 500 large companies listed in the New York Stock Exchange (NYSE) or National Association of Securities Dealers Automated Quotations (NASDAQ) [12]. The S&P 500 is a leading indicator and representation of how well the U.S. stock market does and it is a very liquid index, which means that the slippage due to the difference in the bid and the ask price is very small. One of the reasons why S&P 500 is selected

is that because it is a weighted sum of the 500 stock prices, it is relatively immune to a event in one company such as bankruptcy or merger.

1.2 Problem Statement

The question that this project is trying to answer is whether the historic price of the S&P 500 has any patterns that can be recognized by machine learning algorithms and hence the price of the stock in the past has any indications or clues for the future price of the S&P 500. If so, we can use it as a strategy to trade stocks.

To train machine learning algorithms, historic S&P 500 price publicly available in Yahoo! Finance is used. This data include open, close, adjusted close, high, and low prices, and the volume of trading. Back-testing is done using roll-forward cross validation method. The roll-forward cross validation method is relevant in time series prediction because it splits the data in such a way that train data always come before the test data in the time domain. Hyperparameters are optimized based on the validation accuracy. The hyperparameters that give the best validation accuracy are chosen and different machine learning algorithms with best hyperparameters are compared.

1.3 Metrics

Finally, we would like to trade stocks according to the strategies guided by machine learning algorithms. If the machine learning algorithm predict the stock price would increase based on the opening price, 100 stocks are purchased at the market open and sold at the market close. If it predicts the stock price would decrease, 100 stocks are sold (shorting is allowed) at the market open and bought back at the market close.

The relevant metrics are the return on investment (ROI) and the average accuracy of the prediction for the whole data splits. The ROI is more important in my opinion not only because it is what matters at the end of the day, but it also shows how well the trading algorithm picks up the signal of the stock market increase or decrease.

The benchmark for comparison with the machine learning guided strategy is chosen among the buy-and-hold strategy, random buy-sell, always buy, and always sell strategies. The buy-and-hold strategy is to simply buy 100 stocks of the S&P 500 in the beginning of the testing period and to sell them at the end of the testing period. The random buy-sell strategy is to buy or sell the S&P 500 at random with a coin toss with a half and half probability.

2 Analysis

2.1 Data Exploration

As discussed earlier, the main focus of this project is S&P 500 stock index. Just to emphasize, however, the same analysis used in this project can be used for different stocks. The S&P 500 stock index data can be downloaded from `https://ichart.finance.yahoo.com/table.csv?s=^GSPC`. The data file includes the daily stock price and volume of trading from 1950/01/03 to the current date. The first few lines of data are shown below: The "Open"

Table 2.1: First 5 rows of S&P 500 data

Date	Open	High	Low	Close	Volume	Adj Close
1950-01-03	16.66	16.66	16.66	16.66	1260000.0	16.66
1950-01-04	16.85	16.85	16.85	16.85	1890000.0	16.85
1950-01-05	16.93	16.93	16.93	16.93	2550000.0	16.93
1950-01-06	16.98	16.98	16.98	16.98	2010000.0	16.98
1950-01-09	17.08	17.08	17.08	17.08	2520000.0	17.08

Table 2.2: Summary statistic of S&P 500

	Volume	Adj Close
count	13830	13830
mean	1.045956e+09	622.570068
std	1.606848e+09	612.359619
min	2.020000e+06	52.320000
25%	1.948000e+07	101.660004
50%	1.701200e+08	327.979996
75%	1.422750e+09	1141.465027
max	1.145623e+10	2395.959961

("Close") column is the stock price when the stock market is open (close), the "High" ("Low") is the highest (lowest) stock price in the day. "Adj Close" is the adjusted close price which is the close price adjusted to reflect any distributions and corporate actions such as stock splits, dividend, mergers, and acquisitions.

The problem of this data table, however, is that the open, high, low, and close prices are all the same given the date. This does not make sense because it is not possible that the stock price stays the same all day. Probably this is a mistake in the recording or one price is used for all the columns because other prices are not available.

The statistics of the above data is described in Table 2.2, where the statistics of trading volume and adjusted close price are given. Open, High, Low, Close columns are omitted in Table 2.2 because their values are similar to Adj Close. From Table 2.2, it is clear that numerical value of Volume is much larger than that of Adj Close and the stock price itself is also ranging from ~52 to ~2400. Therefore, to help the convergence of machine learning algorithms, the dataset needs to be normalized. The actual data preprocessing detail is given in Section 3.1.

2.2 Exploratory Visualization

Figure 2.1 (a) shows the S&P 500 historic price from 1962 to 2017. The 120 days moving average was also plotted in the same graph. The 120 days moving average is the average of the stock price of the past 120 days. This technical indicator is normally used for technical analysis to provide the trend of the direction of stock price.

Figure 2.1 (b) shows how much the stock price is deviated from the 120 moving average. For example, if the stock price is 2 standard deviation larger than the 120 moving average, the value in the plot is 2. This quantity is very useful because it shows how extreme the stock price is at the current date compared to the average of past 120 days. This can be more clearly seen from Figure 2.2 (a) and (b) where the stock price from 2000 to 2010 and the deviation from the mean is plotted. We can see that the deviation from the mean graph oscillates approximately between 2 and -2. This indicates that, if stock price rallys to the extreme, it tends to come back to normal or mean. Also notice that the stock price tends to move to more extreme prices to the negative direction than to the positive direction, for example, -3 or -4 standard deviations. This feature is inspired by a technical analysis indicator called Bollinger Bands, which indicates the upper and lower band of stock prices. The upper (lower) band can be 2 (-2) standard deviations [13].

Figure 2.1 (c) shows how much volume of trading happened in particular day with 120 days moving average of volume overlayed on it. Volume also shows oscillatory behavior as shown more clearly in Figure 2.2 (c). When stock price falls quickly, the trading volume tends to increase because the fear in the market increases. In addition, there are periods when volume is very small, which indicates that either complacency of the market is increased or the interest in stock trading is fallen significantly due to sudden increase or decrease of stock prices. These are the features that we might use to predict stock price directions.

2.3 Algorithms and Techniques

In this study, several machine learning algorithms are investigated to see if any of them can predict stock price directions. The machine learning algorithms studied in this project are logistic regression, naive Bayes algorithm, support vector machines, decision trees, random forests, and feed-forward neural networks. The problem is reduced to predict whether the stock price would increase or decrease, which is a classification problem. Therefore, logistic regression instead of linear regression is a reasonable choice. The naive Bayes algorithm assumes that the features are independent of each other which might or might not be true. However, the simplicity of the algorithm works well in practice even though there may be some dependency. The support vector machines and decision trees are very effective when it comes to determine complex boundary for classification even though they have potential danger of over-fitting. The random forest algorithm is used to overcome the over-fitting problem of decision trees. The neural network algorithm is also good at finding complex non-linear boundaries in classification problem. The weights in the neural network are initialized to truncated normal distributions. Drop-out technique is used to fight against over-fitting. Drop-out effectively kills some of neurons at random in the train time and prevent the weights of one neuron from getting too large and over-fitting the data.

Grid search was used to optimize hyperparameters for support vector machines, decision trees, random forest, and neural network. The hyperparameters that give the best prediction accuracy is selected for the final model. Then, the best models in each machine learning algorithm are compared by calculating the return on investment (ROI) when they are used to trade stocks.

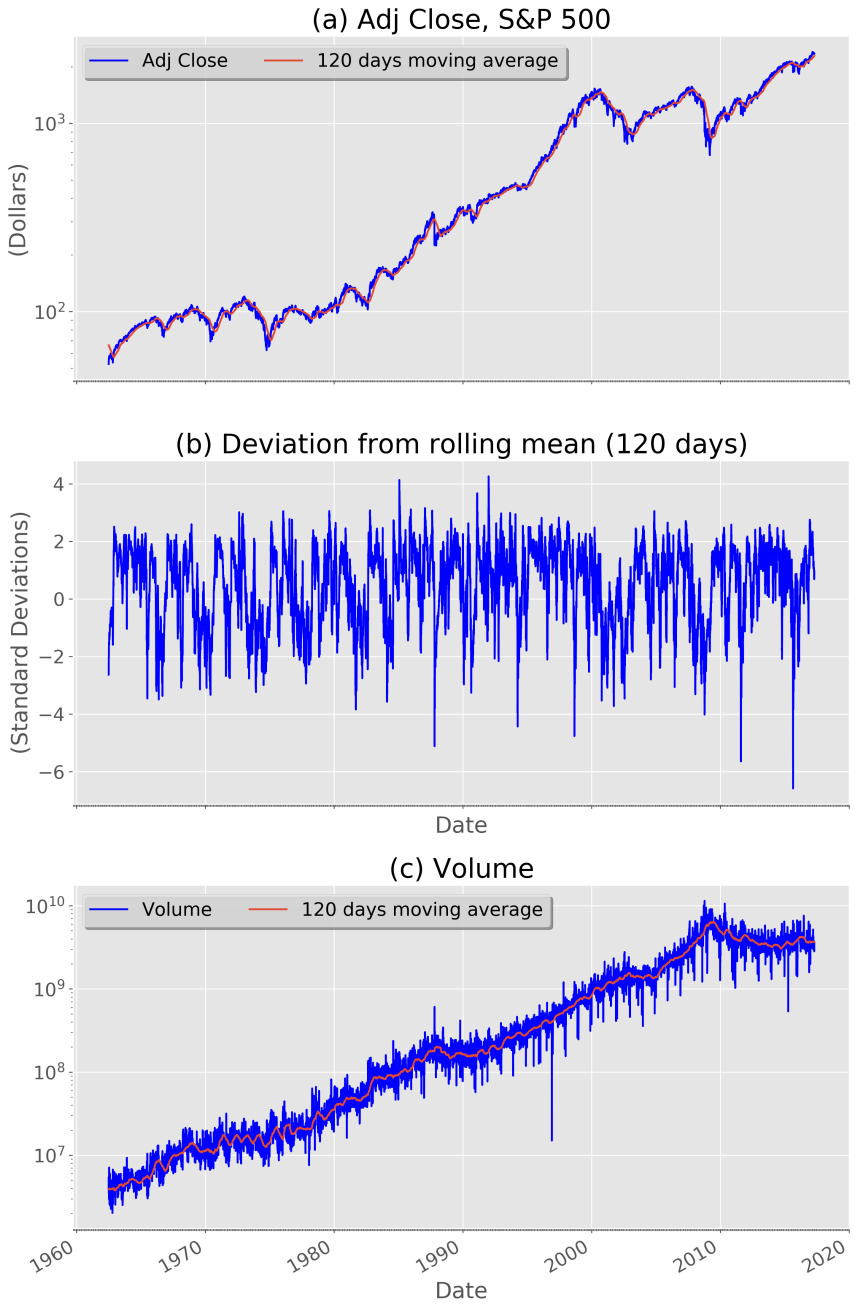

Figure 2.1: (a)S&P 500 historic price, (b)deviation from rolling mean, and (c) volume

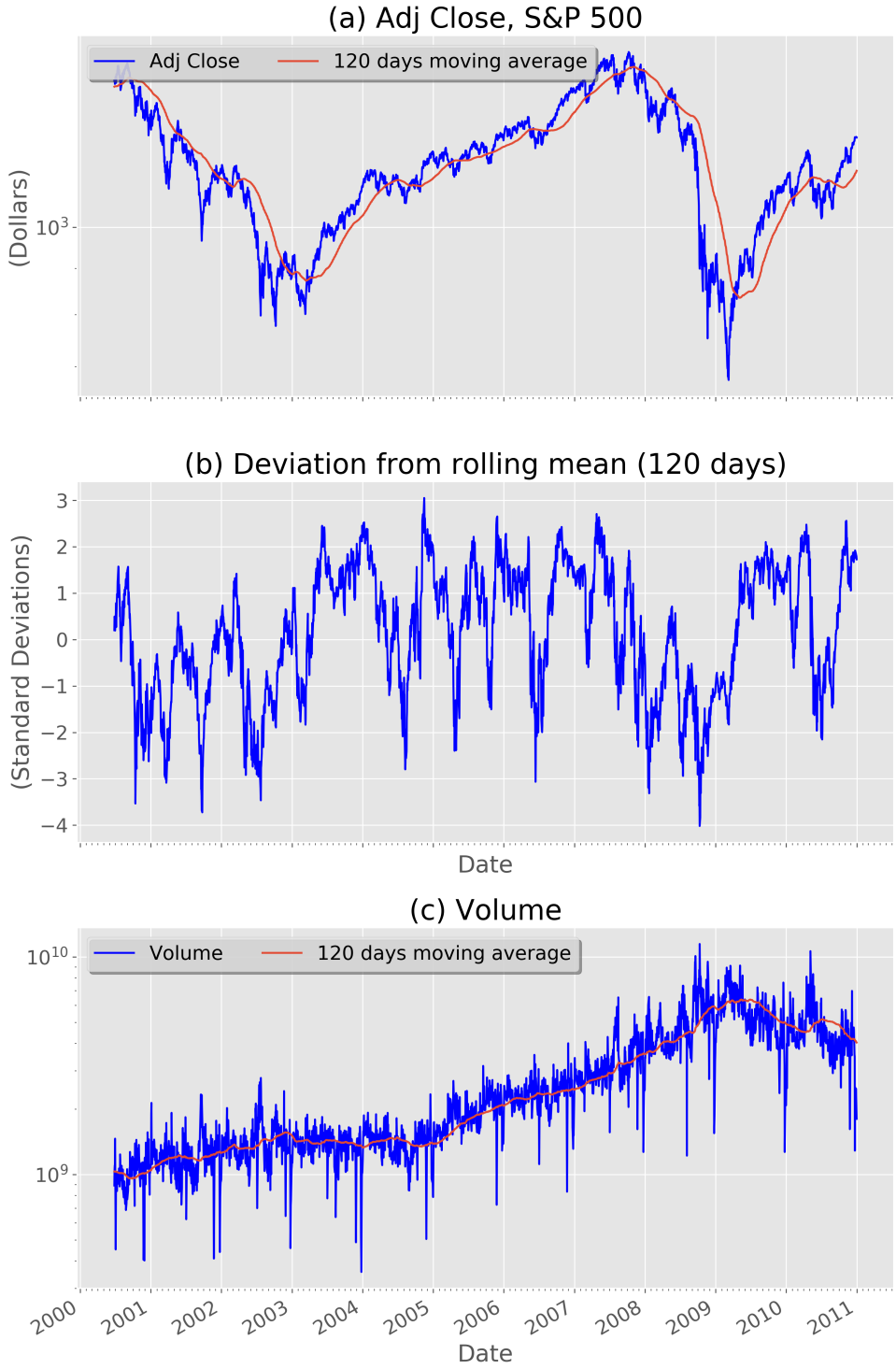

Figure 2.2: (a)S&P 500 historic price, (b)deviation from rolling mean, and (c) volume

Figure 2.3: Benchmark return on investment for different strategy normalized to the initial investment.

2.4 Benchmark

Several different options can be used for the benchmark of trading algorithms. First one is to trade stocks randomly, this means that when the market is open, we flip a coin and buy the stocks if the coin lands on its head and sell them back at the close and if the coin lands on its tail, sell the stocks at the market open and buy them back at the close. The coin flip can be simulated by random number generation. Uniform random number between 0 and 1 can be generated and buy the stocks if the random number is less than 0.5 and sell if more than 0.5. The other option is to only buy the stocks at the market open and sell them at the close or only sell the stocks at the market open and buy them at the close. The final option is to buy the stocks in the beginning of the testing period and hold them until the end of the testing period. Figure 2.3 shows the return on investment when the S&P 500 is traded from 2013-11-27 to 2017-04-21, the same period for which the machine learning is tested. The best option seems to be "buy-and-hold" strategy. Therefore, the benchmark for other machine learning algorithms is chosen to be the "buy-and-hold" strategy.

3 Methodology

3.1 Data Preprocessing

The anomaly found in the data set discussed in Section 2.1 is that the open, high, low, close prices for some periods are all the same in a given day. However, we need the exact values of these prices so that the machine learning algorithm can learn whether the stock price is increased or decreased. In addition, the information of high and low price is important

Date	Open	High	Low	Close	Volume	Adj Close
1962-01-02	71.550003	71.959999	70.709999	70.959999	3120000.0	70.959999
1962-01-03	70.959999	71.480003	70.379997	71.129997	3590000.0	71.129997
1962-01-04	71.129997	71.620003	70.449997	70.639999	4450000.0	70.639999
1962-01-05	70.639999	70.839996	69.349998	69.660004	4630000.0	69.660004
1962-01-08	69.660004	69.839996	68.169998	69.120003	4620000.0	69.120003

Table 3.1: First 5 rows of S&P 500 data after removing abnormal rows

because we can find out how much the stock price is moved and that can be a feature for the machine learning algorithms. Therefore, the rows with same open and close price are removed from the table. The problematic rows are found to appear in the table from the very first date to 1961-12-29. After removing these rows, the data set now shows a correct behavior as shown in Table 3.1.

With anomalies removed from data, the features mentioned in Section 2.2 are calculated and added to the table. The added features are stock price deviation from moving average, volume deviation from moving average, and momentum. In addition, the over-night-return is also added as a feature, which can help the machine learning algorithms to estimate the impacts of the trend overnight on the stock price in the day. The moving average is chosen to be 20, 40, 60, 80, 100, and 120 days. This will give an idea on how much the stock price is deviated from short term to long term average. The momentum is the rate of acceleration of stock price and calculated as [14]:

$$momentum = \frac{close_{today} - close_{N\_days\_ago}}{close_{N\_days\_ago}} \tag{3.1}$$

The momentum is also calculated for 20, 40, 60, 80, 100, and 120 days. The reason these features are selected is that they are easy to understand and widely used for technical analysis.

After the features are calculated and added to the table, the data set is normalized to have zero mean and unit variance when it is fed to the machine learning algorithms for training and testing. This will help the machine learning algorithms to converge faster.

3.2 Implementation

The machine learning algorithms except the neural network are implemented using Scikit-Learn package [15]. APIs provided by Scikit-Learn are used for logistic regression, naive Bayes, support vector machines, decision trees, and random forest. The TimeSeriesSplit function in scikit-learn is used to split time series data for test and training for a roll-forward cross-validation. Table 3.2 shows how the time series data are split in roll-forward cross-validation. The data are split in such a way that the time order of the stock price is maintained. The machine learning algorithm can only see the data in the past to predict the future data. The number of splits chosen is 15. The accuracy_score function in Scikit-Learn is used to calculate the accuracy of testing data for each split, then the average accuracy score is calculated for the whole splits.

Train data splits	Test data splits
1	2
1, 2	3
1, 2, 3	4
1, 2, 3, 4	5
1, 2, 3, 4, 5	6
1, 2, 3, 4, 5, 6	7
...	...

Table 3.2: Traintest data splits for time series data in roll-forward cross-validation. The splits are in chronological order.

The neural network is implemented using Tensorflow [16] which is an open source software for tensor computation. Neural networks can be thought of as computational graphs which represent tensor multiplications and additions. In this project, one, two and three (hidden) layer neural networks are used. For the hidden layers, rectified linear units (ReLU) are used, but activation of the last layer is chosen to be a sigmoid function. The ReLU has advantages of faster convergence for stochastic gradient method and being computationaly efficient [17]. The sigmoid function is used in the last layer to squeeze the output in between 0 and 1 to represent probability. Drop-out function from Tensorflow is used to control the over-fitting. The neural network architecture is shown in Figure. 3.1.

To train the neural network, a mini-batch gradient descent method is used with batch size of 128. The mini-batch gradient descent method is similar to gradient descent method but takes

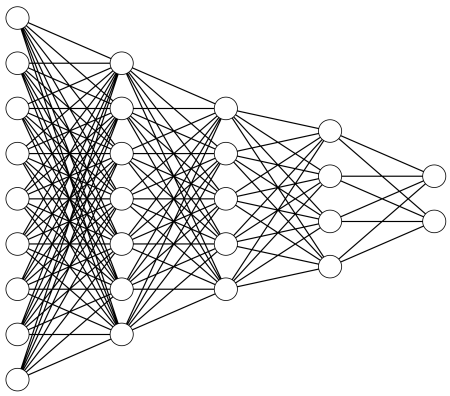

Figure 3.1: Neural network architecture with three hidden layers. The first (last) layer is the input (output) layer.

Machine Learning Algorithm	Initial parameters	Optimized parameters
Decision Tree	min_samples_split=2, criterion=gini, splitter=best	max_depth=1, min_samples_split=2, criterion=gini, splitter=best
Random Forest	max_features=auto, criterion='gini', bootstrap=True, min_samples_leaf=1	max_depth=3, max_features=10, criterion='gini', bootstrap=True, min_samples_leaf=1
Support Vector Machines	C=1, kernel='rbf'	C=1, kernel='rbf'
Neural Network	1 layer, 40 neurons, keep_prob = 0.5, learning_rate = 0.001	1 layer, 1 neuron, keep_prob = 1.0, learning_rate = 0.001

Table 3.3: The model parameters for best models

gradient of subset of data for gradient calculation instead of the whole data set. This way the algorithm converges faster than the batch gradient descent method where the whole data set is used. In addition, Adam gradient descent algorithm [18] is used in the core. Adam gradient descent algorithm is a stochastic gradient algorithm that utilizes second moments of gradients and adaptively selects learning rates. Adam is computationally efficient and does not require large memory. Cross entropy with softmax function is used to calculate the cost function of the output. The target data are one-hot encoded, which means that two outputs, i.e., (1, 0) or (0, 1) are used to indicate that the stock price is increased or decreased. When the neural network algorithm predicts the stock price direction, if the value in the first output neuron is larger than the second, the neural network predicts that the stock price would increase and vice versa.

3.3 Refinement

For support vector machines, decision trees, and random forest, grid search is carried out with the GridSearchCV API provided by Scikit-Learn. GridSearchCV finds the best parameter sets that give the best average accuracy score for all the splits.

For decision tree, max_depth, min_samples_split, criterion, and splitter are used for the parameter search [19]. These parameters are the knobs to tune the complexity of the model. The parameter values are shown in Table 3.3 and the accuracy score and ROI are shown in Table 3.4. The best model chosen by grid search is simplest model in the parameter space. Interestingly, the best model shows less ROI than the initial decision tree model. The reason for this is that the grid search algorithm calculates the accuracy for each split of training data and

Machine Learning Algorithm	Test Accuracy (%)	Optimized Test Accuracy (%)	Return on Investment (initial)	Return on Investment (optimized)
Logistic Regression	53.93		2.71	
Gaussian Naive Bayes	52.62		1.56	
Decision Tree	51.27	55.62	1.40	1.16
Random Forest	55.01	55.03	3.22	2.33
Support Vector Machines	54.25	54.25	2.40	2.40
Neural Network	52.89	56.05	2.24	3.29

Table 3.4: Accuracy score and return on investment (ROI) for different machine learning algorithms

picks up the model that gives the best average accuracy score. However, the ROI is calculated only for the last split which is the testing data set. Therefore, if grid search picks up a model that gives low accuracy score in the last split, but higher accuracy score in the early splits, then it is possible the ROI for the last split might be smaller than one in the early splits and hence the optimized model does not give the best ROI.

For random forest, max_depth, max_features, min_sample_leaf, bootstrap, and criterion are used for grid search [20]. For support vector machines, kernel and C are parameters used for grid search. C is the penalty parameter that determines how accurately the support vector machine classifies the data. Large C means large penalty for wrong classification. These parameters are used to control the bias-variance trade-off [21].

For neural network, since Tensorflow does not provide grid search APIs, a manual search with for-loops is used as a grid search. The number of neurons in the first hidden layer is one of the hyperparameters, but from the second layers, the subsequent number of neurons are chosen to be half of that of the previous layer. There is no hard and fast rules for the number of neurons, but the number of neurons in hidden layers are recommended to be in between that of the layers before and after [22, 23]. The drop-out probability and the learning rate are also hyperparameters optimized to obtain the best neural network architecture which gives the maximum accuracy score.

Initially a neural network with 40 neurons is first tried. The average accuracy 52.89 % and ROI 2.24 is achieved. Then different number of layers, neurons, and learning rates are searched for best accuracy. Interestingly, the neural network architecture that gives the best prediction accuracy is one that is simplest in the hyperparameter search space, that is, 1 hidden layer with 1 neuron. This shows that having more complex machine learning algorithm does not help to improve prediction accuracy.

Figure 4.1: Cost function vs number of iterations for 1 hidden layer with 1 neuron (left) and 4 neurons (right)

4 Results

4.1 Model Evaluation and Validation

Based on the analysis carried out in the previous section, the final model is chosen to be the neural network with 1 hidden layer and 1 neuron. This model seems to be very simple, but we can see why this is indeed the case in the following analysis. First of all, the logical regression can be seen as 1 neuron with sigmoid function and it showed better accuracy and ROI than Gaussian naive Bayes, decision tree, and support vector machines. It is very close match to the random forest algorithm. Therefore we can see that neural network with more than 1 neuron could improve its accuracy than logistic regression. However, having more neurons or hidden layers than the 1 neuron/1 layer did not help improve either the accuracy or the ROI.

Figure 4.1 shows the cost function (cross entropy) for each iteration of the gradient decent algorithm. With 1 neuron in the hidden layer, the training and testing cost decrease and stay more or less constant, but with 4 neurons, the costs decrease until several iterations and start to increase even though the training cost keep decreasing. This shows that the neural network is very sensitive to over-fitting and cannot generalize well as soon as the algorithm becomes more complicated.

This could mean that it is very hard to learn anything from training data with more complex models and that can be due to the changing statistical environment of the stock market with time. In other words, the stock market behavior in the early 1960s might be quite different than that of the 2010s. Therefore, increasing complexity of the model to better fit the old training data does not help to improve testing accuracy of the more recent testing data. Figure 4.2 shows how the accuracy changes with the testing data splits. Early testing data has relatively smaller training data available for training. Therefore, accuracy is also tends to be low. In addition, the testing accuracy in 2000 is lowest among all probably due to the fact that the stock price kept increasing at that time and then long recession happened due to dot-com bubble burst. This leads us to postulate that there are other factors that influenced the stock market. For example, the US government increased supply of base money substantially with quantitative easing [24] and reduced interest rate around 2010 [25]. These are other factors

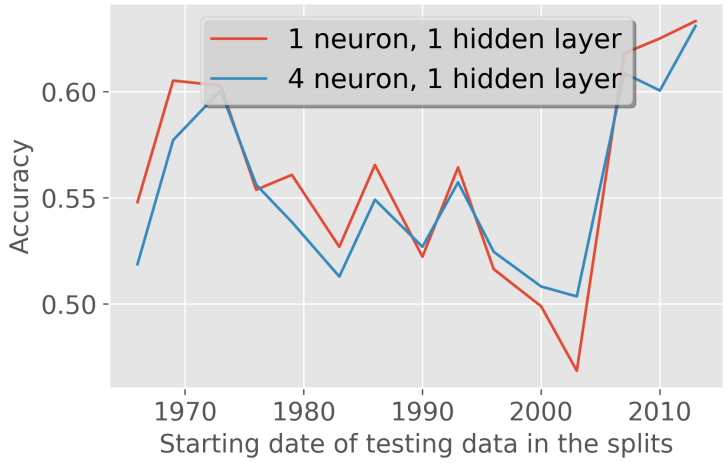

Figure 4.2: Accuracy of testing data vs starting dates of testing data

that influenced the stock market, but they are missing in our data and the accuracy suffers from it.

Figure 4.3 shows the ROI for the last split of testing data set from 2013-11-15 to 2017-04-21. The strategy using the neural network for this period of the time shows a strong ROI with an accuracy of 63.4 %. However, it also shows a poor performance on the data from 2000-04-12 to 2003-09-10 with accuracy of 50.9 % and importantly shows worse ROI than the buy-and-hold strategy. Therefore, it is difficult to say that the algorithm works all the time, but only works better than benchmark at certain times. In other words, the algorithm needs improvement to be used in actual trading.

To test the robustness of the model, I used 10 and 20 train-test splits of data to see how the model performs compared to the original 15 splits. With both of these splits, an average accuracy score 0.54 is achieved. This indicates that the model is somewhat robust to the size of data set.

5 Conclusion

Predicting the S&P 500 stock index price direction with high, low, open, close prices, and volume of trading is a very difficult task. The prediction accuracy on average is larger than 55 % with a simple neural network, but a strategy to buy and sell S&P 500 according to the prediction does not give a consistent ROI for all the testing data. This, on the other hand, gives us a hint that we need more data such as the U.S. interest rate, the base money supply, volatility index - VIX, GDP, labor participation rate, U.S. treasury bond price, and commodity prices to achieve a consistent accuracy and ROI. One can also think of sentiment analysis using publicly available social media such as twitter to provide additional information.

On top of more data, we would need to use more complicated strategy such as reinforcement

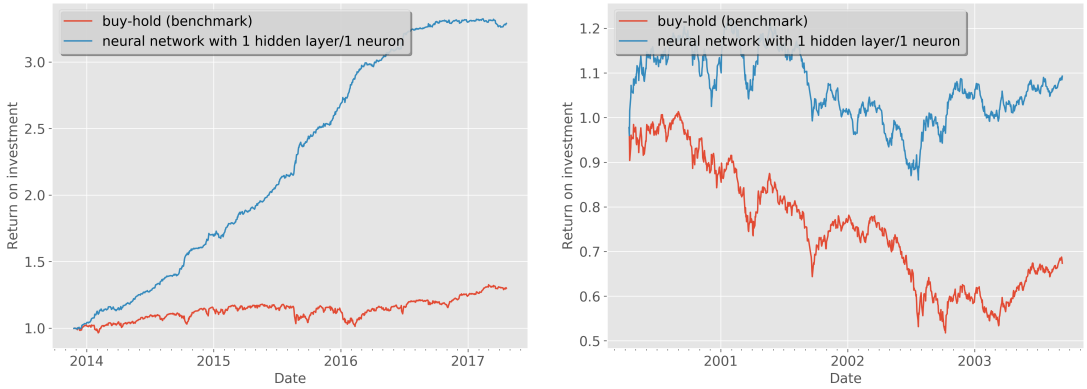

Figure 4.3: Return on investment compared to benchmark from 2013-11-15 to 2017-04-21 (left) and from 2000-04-12 to 2003-09-10 (right)

learning algorithms to buy and sell not only S&P 500, but also bond, or gold ETFs to expand our portfolio. It would be a quite interesting project to work on in the future. In addition, we need to give weights to train-test data splits when we calculate the average accuracy. The accuracy of older testing data should be less weighted than that of latest testing data in averaging the accuracy.

The most challenging part of this project was to optimize hyperparameters in a reasonable time frame with limited computing resources. Optimizing hyperparameters for neural networks takes several hours of runtime using the Intel's latest i7 CPUs. If I made a mistake in the code or a new idea to try, I had to run the optimization again. It would be great if super-computing or GPU computing environment could be used for this kind of project.

If the final solution of this project which is a neural network with one hidden layer and one neuron is the new benchmark, I think there can be a better solution exists if more data are provided. For example, if data mentioned above are given, we could predict not just the direction of stock price (increase or decrease), but how much it would move in the day in terms of percentage movement in discrete categories, say -5, -1, 1, and 5 %. More complex machine learning algorithms such as neural network with more hidden layers or deep learning algorithms such as recurrent neural network algorithms can help to enhance the accuracy of the prediction.

REFERENCES

[1] A. C. Andersen, *A Novel Algorithmic Trading Framework Applying Evolution and Machine Learning for Portfolio Optimization*. PhD thesis, Norwegian University of Science and Technology, 2012.

[2] Wikipedia, "Efficient-market hypothesis." `https://en.wikipedia.org/wiki/Efficient-market_hypothesis`.

[3] D. Berreby, "Chaos hits wall street," *DISCOVER-NEW YORK-*, vol. 14, pp. 76–76, 1993.

[4] M. G. Kendall and A. B. Hill, "The analysis of economic time-series-part i: Prices," *Journal of the Royal Statistical Society. Series A (General)*, vol. 116, no. 1, pp. 11–34, 1953.

[5] A. W. Lo and A. C. MacKinlay, "Stock market prices do not follow random walks: Evidence from a simple specification test," *Review of financial studies*, vol. 1, no. 1, pp. 41–66, 1988.

[6] M. D. McKenzie, "Chaotic behavior in national stock market indices: New evidence from the close returns test," *Global Finance Journal*, vol. 12, no. 1, pp. 35–53, 2001.

[7] A. Bensaïda and H. Litimi, "High level chaos in the exchange and index markets," *Chaos, Solitons & Fractals*, vol. 54, pp. 90–95, 2013.

[8] K.-j. Kim, "Financial time series forecasting using support vector machines," *Neurocomputing*, vol. 55, no. 1, pp. 307–319, 2003.

[9] R. Choudhry and K. Garg, "A hybrid machine learning system for stock market forecasting," *World Academy of Science, Engineering and Technology*, vol. 39, no. 3, pp. 315–318, 2008.

[10] T. Kimoto, K. Asakawa, M. Yoda, and M. Takeoka, "Stock market prediction system with modular neural networks," in *Neural Networks, 1990., 1990 IJCNN International Joint Conference on*, pp. 1–6, IEEE, 1990.

[11] Y. Zhang and L. Wu, "Stock market prediction of s&p 500 via combination of improved bco approach and bp neural network," *Expert systems with applications*, vol. 36, no. 5, pp. 8849–8854, 2009.

[12] Wikipedia, "S&p 500." `https://en.wikipedia.org/wiki/S%26P_500_Index`.

[13] J. Bollinger, "Using bollinger bands," *Stocks & Commodities*, vol. 10, no. 2, pp. 47–51, 1992.

[14] Wikipedia, "Momentum (technical analysis)." `https://en.wikipedia.org/wiki/Momentum_(technical_analysis)`.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[16] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[17] "Cs231n convolutional neural networks for visual recognition." http://cs231n.github.io/neural-networks-1/, author =.

[18] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[19] Scikit-Learn, "sklearn.tree.decisiontreeclassifier." http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html.

[20] Scikit-Learn, "sklearn.ensemble.randomforestclassifier." http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html.

[21] Wikipedia, "Bias-variance tradeoff." https://en.wikipedia.org/wiki/Bias%E2%80%93variance_tradeoff.

[22] comp.ai.neural nets, "Section - how many hidden units should i use?." http://www.faqs.org/faqs/ai-faq/neural-nets/part3/section-10.html.

[23] Naved, "How to choose the number of hidden layers and nodes in a feedforward neural network?." https://iqbalnaved.wordpress.com/2016/12/26/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-network/.

[24] Wikipedia, "Quantitative easing." https://en.wikipedia.org/wiki/Quantitative_easing.

[25] T. Economics, "United states fed funds rate." http://www.tradingeconomics.com/united-states/interest-rate.