Feasibility of Using Machine Learning Algorithms to Determine Future Price Points of Stocks

# Introduction

The stock market is considered by many people little better than gambling. However, if you have the right edge, it can be hugely rewarding. However, this requires the right edge. There are many financiers in Wall Street making a fortune while we plebeians suffer their consequence. These stock market traders have one elemental advantage that they can afford to pay for these “black boxes”, a colloquial term for pieces of programming that accurately predicts the stock prices in the future, and automatically makes the trades based on experience or expertise to the order of thousands of transactions per milliseconds. Armed with these tools, these people can make a fortune due to being able to “predict” future stock prices. In fact, machine learning is grabbing the attention of the big financial magazines, such as Bloomberg and Wall Street Journal ([1], [2]). The goal of this project is to figure out if machine learning will realistically produce an advantage in the stock market world, and what algorithm is better. This paper will be driven by looking at what other people have done, and applying it to the NASDAQ list of companies with stock data for 2015.

# SQL Database

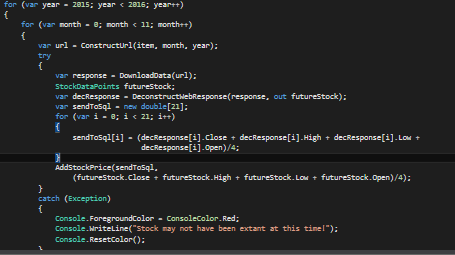
We used the YAHOO API for acquiring stock information. However, this solution is limited in that it only provides data to day resolution at the most. For the day, we could have the day’s open for that stock, day’s low, day’s high and day’s closing price. To keep things as simple as possible, we averaged all four of those prices to produce a general average price for that day’s trading.

|  |  |  |
| --- | --- | --- |
|  |  | 1 |

This does not take into account dramatic changes for the day, but seems a reasonable estimate of the prices for that day. We send a download request to YAHOO API to return 2 months of stock data at a time. We could have made it more efficient by requesting the whole 2015 data for that stock and slitting up that way, however, we were not certain if all the stocks on the NASDAQ list existed in 2015, or were founded in late 2015, so we determined that gathering stock information in two months segment was the safer bet, whilst being slower. We collected the first 21 business days, representing approximately a month, (), and we also collected the price at the end of the two month streak. We then calculated the future price as a ratio of the current average of all 21 days to the price at the end of two months. This took the form:

|  |  |  |
| --- | --- | --- |
|  |  | 2 |

Where 1 is a strong buy, 2 is a weak buy, 3 is do nothing, 4 is a weak sell, and 5 is a strong sell. These are the effective labels based on the future price compared to the average current price. The ratios are arbitrary and can be anything we want. We parse the return data from YAHOO API, as a string containing all the information we need split up by characters. Once we have the data, we send it to a local SQL database as 23 numbers, the first 21 prices, the future price, and the integer that represents the position of that stock. The stock company list from NASDAQ contained companies that did not necessarily exist in 2015, so we encapsulated the download method to the YAHOO API with a try…catch block, so as to stop the program crashing in the case that a company did not exist at that time. We also only chose to get information on all the companies in 2015. We could have gone back as far as 1980 but then companies like MICROSOFT and APPLE would have taken precedence over companies that were founded much more recently, and the asymmetry would have produced skewed results. The final result was a table with 32547 representative stock data listings (. We also could have realistically done a sliding window, which would produce stocks with , where n is the window size, and h and w are holidays and weekends. However, this would have produced a huge number of redundancies so we decided not to attempt it. We also do not collect December data because we were being exceedingly lazy and didn’t feel like coding an if…else to switch the year onto the parsing line.



Parsing the data into SQL means we could have a generic line of code that could access the database, the data was returned in a second as opposed to using a WebClient to extract the information from the YAHOO API, which took about 10 minutes for 3000 stocks. If using the code samples provided with this paper, you will need to run SqlSaver once before going to the other sections of the code.

# Website Parser

To get this out of the way, we produced a stock website parser. The stock parser checked 12 major financial websites (WSJ, Bloomberg, Investopedia, CNNmoney, etc…) and extracted phrases from these websites. It then checked if those phrases had any negative words in it by comparing it to the negative words in Loughran McDonald’s master dictionary and if it had a stock ticker from the NASDAQ list. The idea was to check if that phrase contained any negative connotation for that particular NASDAQ ticker. If the phrase did not contain a negative mention for that particular stock ticker, then it checked if that phrase had a positive word from the McDonald’s list, so that the only outputted stocks were ones that occurred in phrases that had no negative mentions and had a positive mention. An example output is seen below:



We can also get the web parser to print out the sentences associated with these positive tickers. This is supposed to complement the machine learning algorithms we tested.

# Naïve Bayes Algorithm

The first algorithm we tested was the Naïve Bayes algorithm[3, 4]. Naïve Bayes algorithm is based on Baye’s Rule, that is, for particular data and variable , Baye’s Rule says that we can update our beliefs according to:

|  |  |  |
| --- | --- | --- |
|  |  | 3 |
|  |  |  |

Based on evidence received, we update this data generating model. tells us what we can expect the probability of something happening is prior to the evidence proposed, is the evidence proposed. We can simplify the model for explanations sake. The example is taken from khan academy: Bob is in a room, he has two coins, one is a fair coin that flips to heads or tails randomly, and the other coin is double-sided and has two heads. He picks one randomly ad we then construct a tree representation. He could have realistically picked up a fair or unfair coin, so there are two branches, one that goes to the fair coin and one that goes to the unfair coin. He then flips the coin he extracted from the bag, so the tree grows further. IF it was the unfair coin, then both outcomes would be head, if it was the fair coin there is an equally good chance that heads or tails would appear. Then the evidence factor comes into play. Bob says heads was his outcome, so the tree is pruned of every possibility that did not end up with heads. The probability that he attained a fair coin that displayed heads was thus equal to . He then flips a new random coin. The tree grows to include this new outcome. On the unfair side of things, the probability of getting heads is 100%. On the fair side of things, the probability of getting heads again is equal to The probability of getting a fair head in this case is since there are five total heads outcomes and 1 true heads outcomes. The algorithm works in the same way, updating the likelihood after each new piece of evidence is presented. However, something is not correct in our case. The naïve Bayesian design of our model closely resembles that which was introduced in [5]. They state that Naïve Bayes is a strong classifier when the features collection is high, such as in our case, however, it assumes naivety, or independence, between the features set. In our case, can we really do this? Is day 2 opening price not based slightly on day 1 closing price. They are somewhat related but we will still perform the analysis. The overall formula quoted in this paper is:

|  |  |  |
| --- | --- | --- |
|  |  | 4 |

So that the current statistical model is based on the evidence of how many features there are and the previous likelihood. Bafandeh concerned himself with 3 classifiers, we have two more classifiers so more resolution in classifiers. To summarize what we have up until now: 21 stock prices, a future stock prices, and a classifier for that stock price that relates growth. The Naïve Bayes algorithm is provided by ACCORD for the .NET environment. We input all the data from SQL into the algorithm and then run it back on all the data to test for accuracy. The error is judged as RMS (root-mean-squared). After the completion of the Naïve Bayes run, the error was outputted and turned out to be , which is a horrendous error on 5 classifiers. Assuming the Naïve Bayesians algorithm tells us Do Nothing, it could realistically mean Strong Sell or Strong Buy. This classifier is not very tolerant unlike what other people have said [3, 5]. However, other research has complimented this with other external classifiers such as 10-day moving average and Commodity Channel Index.

# K Nearest Neighbor

K nearest neighbor is one of the simplest machine learning algorithms. Since there are 21 features in our case, the K nearest neighbor algorithm will try to classify each feature to the closest classification. In our case, there are five distinct classification and 21 features. kNN is a statistical technique that aims to classify data in its similarity to a neighbor. [6] did work relating to kNN algorithms as it relates to stock predictions. KNN is easy to apply and does not produce a model but rather stacks the features to the closest neighbors by some metric, usually Euclidean distance. In [6]’s case, they applied the kNN algorithm to a very small group of stocks with 200 days of high, low and close information associated with each stock in a sliding window. They compared the predicted and actual values and had an overall RMS of 0.0378 between the prices, so this method should be pretty good. When we run kNN with the top 5 neighbors in consideration, the RMS error between our expected and actual classifier is just 0.64. When we increase the k nearest neighbors to 10, the error goes down to 0.62 and when we do 128 nearest neighbors, we get 0.55 RMS error. We did an 80/20 split of training and testing data on the full SQL database. The accuracy was determined as number of correct outputs compared to the all outputs. RMS suggested that there is decreased error with the use of more neighbors. The accuracy, that is, the number of correct predictions also increased with increasing k’s.

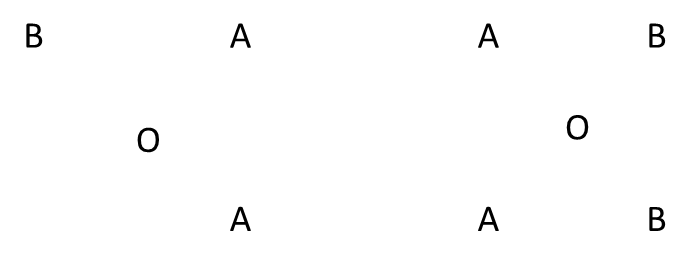
|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

The number k decides how many neighbors influence the classification. In the case that k=1, then if we have some point x, with y being the closest labeled point to x, then x will take the label of y. This technique is based on size. We luckily have about 32000 data points, so we’re pretty safe. If we only had 100 points, it would be a different matter. For multi-dimensional or multi-featured nearest neighbor, the reasoning is exactly the same. Neighbors farther away from the point x are less likely to be chosen and ones closer are more likely to be chosen. This particular algorithm surprisingly was relatively accurate at low k’s, which is odd as usually accuracy improves with increasing k’s[7], as does computation.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

Trying different distance metrics changed the RMS error slightly.

Usually, the kNN algorithm takes a majority vote to place a label on an observation. So if k = 4, 3 of the classifiers belong to class y and 1 classifier belongs to class x, kNN will assign the observation to classifier y. However, what if k is even, and there are two classifiers for x and y, and they are equidistant from our observation so that weights don’t come into play, which one will kNN choose for our observation? This can be avoided by carefully choosing k so this doesn’t occur.



In a two dimensional plane, if k = 3, like in the left hand picture, then the kNN algorithm can confidently say that from the 3 nearest neighbors, observation O belongs to class A. However, when k = 4, like in the right example, it will try to set weights to each edge. In this case, however, all the weights are exactly the same since the classifiers are equidistant from O, then kNN will have to try some conflict resolution procedure to determine if O should be in A or B.

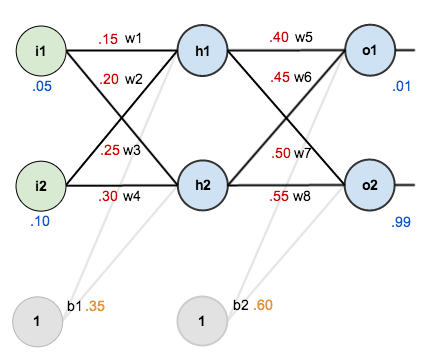
# Artificial Neural Networks (ANN)

One modelling strategy that seems popular in the stock market world is artificial neural networks, from now on referred to as ANN. The ANN is different from the previous two methods in that it tries to make a model to explain the inputs it is given. ANN models, at the simplest, consist of an input layer, an output layer and several hidden layers. The inputs are passed through the input layer, assigned a weight, and forwarded to one of the hidden layers, where it is assigned another weight, and eventually reaches the output layer. The idea is over time, with or without back-propagation, the model will produce a set of weights that will optimize the problem at hand. [8] proposed to analyze the Spanish stock market for General Index of Madrid using a simple neural network that took in 9 days of stock prices to estimate the future price. Their model was simply:

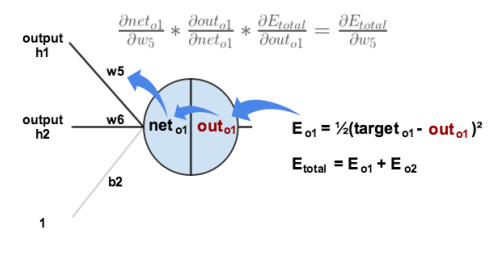
|  |  |  |
| --- | --- | --- |
|  |  | 5 |

Which represents one hidden layer with four hidden neurons and one output layer. Their overall conclusion was that ANN provided an edge for stock market trading.

Back-propagation takes place after each new piece of evidence is presented to update all the weights and bias, so that the model stays as up to date as possible. The ANN model can be feed-forward or back-propagation ready. The Spaniard just did his paper with a feed-forward algorithm, whereas we use back-propagation to minimize the error in the neural network after each pass of data. We shall consider the back-propagation given in Matt Mazur article “A step by Step Backpropagation Example”, to explain backpropagation.



We first pass forward through the hidden layer, from inputs hidden neurons output. At each node of our chart, we calculate the net input . The input to the hidden neurons are then transformed as output by and the net output’s input at o1 becomes and the output becomes .We then calculate the error from the actual and expected output, as . Up until now, this is simply a feed-forward model, much like equation 5 which Fernandez-Rodrıgueza implemented whilst trying to predict the General Index of Madrid. Now we shall do a backwards pass to each weight. For example, we shall consider in the example model.



The total error changes as a function of , and we want to minimize this error. The total error formula is The partial derivative in terms of is , the partial of in terms of net O1 is and the partial of in terms of is so the total error due to is can now be calculated as . We then update the weights as . We repeat this process for all weights. In this case, is considered to be the learning rate of the system. [9]

In our case, we followed a similar model, except we could include or exclude epochs. The algorithm we used was based on McCaffrey’s excellent article on the subject[10]. In this case, we split our database into two segments; training and testing data. We experimented with different ratios of testing and training data, and the results are summarized in the table below. The number of inputs was 21 stock prices for 21 consecutive trading days, the output was the classifier, and we provided a single layer of 4 hidden neurons. The error we got from this particular model in both the training and testing set was 56% and 57% respectively. The error in the training set is useful for determining if there was model-overfitting, and the accuracy of the test data tells us how good the model fit was.

|  |  |  |
| --- | --- | --- |
|  | | |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

ANN has shown to be a powerful tool for predicting future stock prices, although it suffers from computational complexity and lack of efficiency; we have to run many more epochs to get more accurate results, which takes much longer.

We also ran the back-propagation parallel resilient algorithm provided by ACCORD. The implementation of this algorithm can be found at (<http://accord-framework.net/docs/html/T_Accord_Neuro_Learning_ParallelResilientBackpropagationLearning.htm>). The only thing we changed was the outputs and inputs, as well as associated numbers for hidden neurons and input and output layers and activation functions.

|  |  |  |
| --- | --- | --- |
|  | | |
|  |  |
|  |  |
|  |  |
|  |  |

This particular machine learning algorithm does not work terribly well.

# Parallel Resilient Back-Propagation Learning

In our everlasting quest to find that particular algorithm that perfectly describes a stock’s future price by prediction, we find ourselves going into steadily more complex algorithm, as we can tell by the names given to them (12 syllables for this one). Once again, James McCaffrey comes to the rescue, defining this complex algorithm in simple words we can all understand. Rprop, as it shall now be called, has two distinctive advantages over regular back-propagation presented above. It is faster than typical back-propagation, and it doesn’t need hard coded values for momentum and learning rate. Much like with ANN, Rprop accepts a layer of inputs, passes them through a hidden layer, and then passes them to an output layer. However, ANN is highly sensitive to the learning value input by the user. Rprop was implemented as a possible solution to this problem in 1993.[11] The Rprop algorithm provided by McCaffrey finds the optimal weights and bias between the hidden, input and output layers that minimizes the error between the inputs and outputs in terms of learning rate and momentum. The Rprop algorithm is an iterative process that processes multiple epochs, going back on itself for how many time the user specifies. McCaffrey was able to get 98.75 percent accuracy on random data with his algorithm, which we have adapted to work with our data.

Using the same data and the same 80/20 ratio we used in ANN, we were able to compute a training and testing data of 42% and 41% respectively. The final computed error after 1000 epochs was 0.69 RMS, which is significant. Finding a good compromise between accuracy and computational time is considered in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

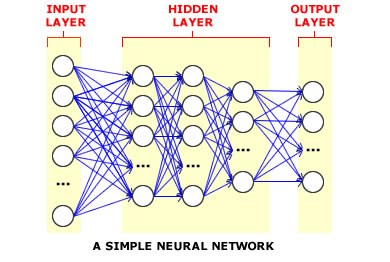
We also investigate the ratios of training to test data to see how the accuracy changes.

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

The problem in our cases is that the stock market truly is random at times, it can be influenced by the day’s weather, socio-political issues, scandals, sex tapes, and so on, and we realistically have no way of predicting these factors by just using price points. Thus the class identifiers for the future price as it relates to the current price is truly random, so no model can hope to fit this random walk, which is why ANN and Rprop both failed to make an accurate model.

# Deep Neural Network Learning

Deep neural learning functions exactly the same way as ANN does, except that instead of having a single hidden layer, it can have several hidden layers. The mathematics and concept behind the model stay exactly the same though, even though the complexity increases.



We ran the Deep Learning Algorithm for a 60/40 slit of training and testing data with varied epochs and hidden layers; the results are presented below:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

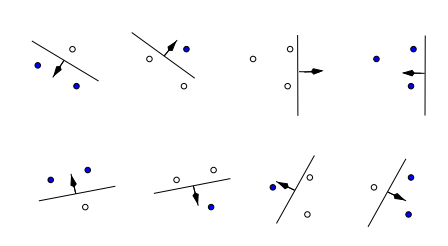
It seems that the model is overfitted as it returns the same % accuracy on the testing data, and in fact, on closer inspection, it returns the exact same cases as true regardless of number of hidden layers, epochs run. So it seems that this model is not suitable for stock market predictions.

# C45 Learning Algorithm

Now we enter the algorithm made by J. Ross Quinlan, C45 Learning Algorithm. This is a decision tree formatted algorithm that is statistically inclined. [12] Decision trees are quite good at dealing with numerical data and can be used as a statistical classifier like Naïve Bayes. Quinlan’s algorithm C45 became quite popular after it featured in ‘Top 10 Algorithms in Data Mining’ published by Springer publication in 2008. C45 uses the concept of information entropy to sort features with classification . The algorithm subsequently chooses the sub-feature with the highest information entropy to make the decision, so it’s a dimensionality reduction of sorts. It does this for all the data, and builds up statistics about the most likely outcome for particular features[13]. The difference from ID3 algorithm is mostly that C45 will try to minimize the final tree so that ineffective branches on the tree are pruned off. For our ~32000 data sets, the algorithm took 49 minutes to run, and the RMS error was 1.09, so not that good and the C45 algorithm correctly guessed 17026 of 32457 (52% accuracy)stock trends, which is a bit better than half.

# Support Vector Machine

Support Vectors Machine are best explained by [14], and I can just summarize how he explains them. There are many observations, in our case, approximately 32000. Each observation contains a pair of data; from 1 to l, and an associated truth value from 1 to l. A particular machine has the job of assigning a mapping for each , however, it will assign a probability distribution instead: where is an adjustable parameter. The set is then shattered in Vapnik Chervonenkis dimensions: which is displayed graphically below:



For simplification, we assume that is composed of straight lines to make things easy and we assume that we have two features we are inspecting so that the dimensionality is **.** All the points on one side of the line equals -1 and all points to the other side equals 1. For different lines, there are different numbers of 1 and -1 classified sets. The arrow points to the direction of label 1. The point of this is to generalize that for any there is only n+1 VC’s. This is based on linear independence; we can choose one point as being the origin, and the other n points as linearly independent points. However, introducing more points on the set are not guaranteed to be linearly independent past n+1. The point of SVM is to choose the that will have minimal VC dimensions associated with it.

This can be best explained as the line that crosses the plane should be as far from all the points we have. SVM does this by associating a margin with each line it tries, and chooses the line with the largest margin. [15] We consider a line with formula . There are an infinite amount of lines that could span the null space corresponding to our data points. The optimal hyperplane (k-1 from dimension) for these points can be normalized so that with x in this case being the training data closest to the hyperplane. The distance between the hyperplane and the closest points to the hyperplane are a distance of which for the optimal hyperplane is equal to since . We need to maximize the distance of the margin, which we can do by minimizing some function subject to for all x in our set.

First, we tried ACCORD’s SVN. However, the SVN in this case really hogged the memory and gave a “OutOfMemoryException” with 8 GB of memory installed on my computer. We therefore reduced the input and output sets to 5000 observations, so as to get a result. Even at 5000 observations, the algorithm consumed 830 MB of memory. The returned RMS error was 0.53 and the algorithm correctly guessed 2660 out of 5000 observations, which represents a 53.2% accuracy on three labels, consistent with Kim et al[16]. Running the SVM on the first 5000 observations and testing it on the next 5000 observations to simulate a 50/50 split of training and testing data produced an RMS error of 1.29 and correctly guessed 2891 of 5000 observations, so an accuracy of 57.8%, again consistent with the findings of Kim et al. Running the algorithm on the next 5000 testing data produced RMS error of 1.25 and correctly guessed 3029 of 5000 classifiers, 60%.

We re-ran the algorithm implementation of SVM with the core library provided by LIBSVM.NET which could handle the 32000 stock observations. We split the stocks up in 50/50 training and testing sets. The algorithm took 12:40 to run and had an RMS of 1.26 and accuracy 60%.

# Discussion

We have established that no model based algorithm will produce a model of our stock prices that will return the classifiers we expect with great accuracy, due to the chaos in our system. This is discouraging news but we may now consider two different options; change the parameters we input to be modeled that are a little bit more consistent, or consider other algorithms. Both of these options will take a lot of work, but we shall consider the change of parameters first. It seems our model of 21 prices and a future price is not so good, and the classifier model of 5 classifiers is also not so good. We try to re-create what [8] did. He considered nine previous prices and the output is equal to 3 classifiers (-1,0,1) where 1 represents buy and -1 represents sell. Just reducing the classifier from 53 produced vastly superior results, from ~44% accuracy on 5 classifiers on training and testing data, to 90% and 89% on training and testing data respectively at 100 epochs run with Rprop. When the inputs are reduced from 219 inputs, like in [8], the accuracy stays more or less the same at 90% and 89% accuracy respectively, which supports Rodriguez’s statement that the number of inputs above 9 produced marginally the same qualitative results. Increasing the amounts of epochs run iteratively on the solution to 10000 produced an accuracy of 91% and 90% respectively for training and testing data, although at a computational cost of almost 13 minutes. We lost some resolution in the classifiers but gained a lot more accuracy in the process, which I think is much more important. Reducing the classifiers to just a binary state of 1 and 0 to represent buy or sell produced the same results as using 3 classifiers. So it seems that the ideal situation is a lower input strand and a reduced classifier output to produce better results. Switching the Naïve Bayes algorithm to function with 3 classifiers as opposed to 5 classifiers also improved the RMS error to 0.74 and the accuracy achieved was 44.9%.

# Conclusion

Experts in the field, such as [17] and [16] have tried to fit the stock market trends with greater accuracy using algorithmic hybrids that take advantage of multiple algorithms, yet they achieved more or less the same numbers as we did, even though their algorithms were much more elaborate in certain cases. Through our review of machine learning algorithms, we conclude that none of the current algorithms in this paper, whether it models the data or not, is adequate as a standalone to make accurate decision on the stock market. Maybe having data sets with more resolution could influence potential profit to be made with these models. The key thing to take into account is that if someone is publishing an article on his or her algorithm, it probably does not work that well. The successful people take their successful algorithm and start a company.

# References

1. Bit, K. *The $10 Hedge Fund Supercomputer That’s Sweeping Wall Street*. 2015.

2. *Natural Language Processing, Machine Learning Power Staples Ordering System*. 2016

3. Taylor, B., *Applying Machine Learning to Stock Market Trading*.

4. Barber, D., *Bayesian Reasoning and Machine Learning*. 2012, Cambridge Univ Press: Univ Press, Cambridge.

5. Imandoust, S.B. and M. Bolandraftar, *Forecasting the direction of stock market index movement using*

*three data mining techniques: the case of Tehran Stock Exchange*. 2014.

6. Alkhatib, K., et al., *Stock Price Prediction Using K-Nearest Neighbor (kNN) Algorithm*.

7. Thirumuruganathan, S., *A Detailed Introduction to K-Nearest Neighbor (KNN) Algorithm*. 2010.

8. Fernandez-Rodrıgueza, F., C. Gonzalez-Martela, and S. Sosvilla-Riverob, *On the profitability of technical trading rules based on artificial neural networks: Evidence from the Madrid stock market*2000, Economics Letters.

9. Mazur, M., *A Step by Step Backpropagation Example*.

10. McCaffrey, J., *Coding Neural Network Back-Propagation Using C#*. 2015.

11. McCaffrey, J., *How To Use Resilient Back Propagation To Train Neural Networks*. 2015.

12. Rokach, L., *DECISION TREES*.

13. *C4.5 algorithm*. Wikipedia.

14. BURGES, C.J.C., *A Tutorial on Support Vector Machines for Pattern*

*Recognition*. 1998.

15. *Introduction to Support Vector Machines*. OpenCV.

16. Kim, K., *Financial time series forecasting using support vector machines*. 2003, Neurocomputing.

17. Wang, Y., *Market Index and Stock Price Direction Prediction using Machine*

*Learning Techniques: An empirical study on the KOSPI and HSI*. 2013