Boosting and Bagging of Neural Networks with Applications to Financial Time Series

Zhuo Zheng

August 4, 2006

Abstract

Boosting and bagging are two techniques for improving the performance of learning algorithms. Both techniques have been successfully used in machine learning to improve the performance of classification algorithms such as decision trees, neural networks.

In this paper, we focus on the use of feedforward back propagation neural networks for time series classification problems. We apply boosting and bagging with neural networks as base classifiers, as well as support vector machines and logistic regression models, to binary prediction problems with financial time series data. For boosting, we use a modified boosting algorithm that does not require a weak learner as the base classifier.

A comparison of our results suggest that our boosting and bagging techniques greatly outperform support vector machines and logistic regression models for this problem. The results also show that our

techniques can reduce the prediction variance. Furthermore, we evaluate our model on several stocks and indices using a trading strategy, and we are able to obtain very significant return on investment than the market growth rate.

1 Introduction

Data mining is the process of analyzing large quantities of data and summarizing it into useful information. Supervised data mining considers training a model from training data set, which will enable us to make out-of-sample predictions. New techniques and algorithms have been created in practice as development of powerful computer processors allows for increasingly complex computations.

Boosting is a technique to improve the performance of the machine learning algorithms. Boosting combines ”weak learners” to find a highly accurate classifier or better fit for the training set (Schapire, 1990). Successive ”learners” focus more on the errors that the previous ”learner” makes. Bootstrap aggregating (bagging) is another technique designed to improve the performance of machine learning algorithms(Breiman, 1994). Bagging combines a large number of ”learners”, where each ”learner” uses a bootstrap sample of the original training set.

Both boosting and bagging have been successfully used in machine learning to improve the performance of classification algorithms such as deci-

sion trees. There have been a number of studies on the advantages of decision trees relative to neural networks for specific data sets and it has been shown that boosting works as well or better for neural networks than for decision trees(Schwenk and Bengio, 2000).

In this paper, we will investigate the predictability of daily financial time series directional movement using a data mining technique. We focus on the use of feedforward back propagation neural networks for time series binary classification problems. We apply boosting and bagging with neural networks as base classifiers, as well as support vector machines (SVM) and logistic regression models, to binary prediction problems with financial time series data such as predicting the daily movements of stocks and stock indices. For boosting, we use a modified boosting algorithm that does not require a weak learner as the base classifier.

Three experiments are designed to evaluate the performance of our techniques and model. First, comparing the percentage of the directional success of our models to the SVM and logistic regression. Second, examining the statistical performance of the models, such as increasing accuracy and reducing variance. Third, applying a trading strategy to our model output and determining the return on investment compared to the actual market growth.

2 Learning Methods

2.1 Neural Network

A neural network illustrated in Figure 1 is a general statistical model with a large number of parameters.

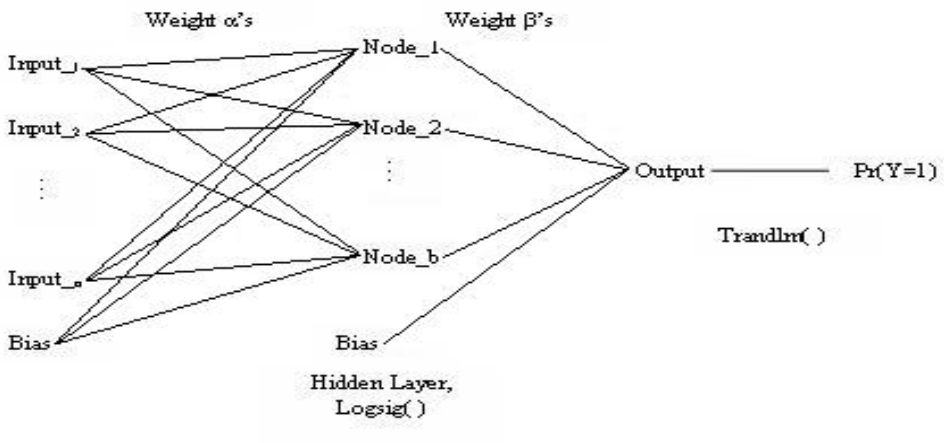

Figure 1: Neural network with weight parameters and transform function.

A feedforward back propagation neural network trains all the training data (or example) repeatedly with difference weights. A black-box like neural network trains data as shown in figure 2. Neural networks have been trained to perform complex functions for regression and classification. For a binary classifier of classes A and B, a two-node output will return probabilities of being classified as class A or class B. Since $Pr(A) + Pr(B) = 1$, we will build the neural network to return P(A), and obtain Pr(B)= 1-Pr(A). The architecture of the feedforward back propagation neural network is de-

noted "$a-b-1$" with 1 hidden layer, where a is number of elements in the input vector, and b is the number of the nodes in the hidden layer.

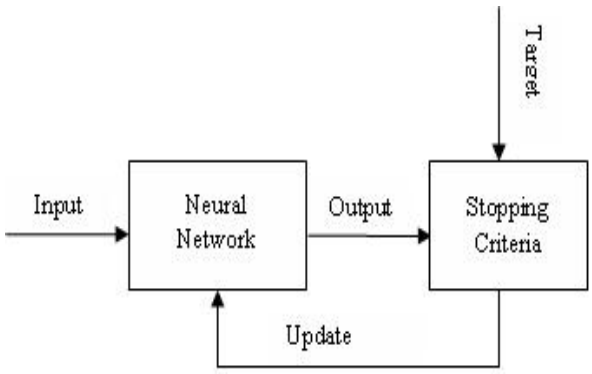

Figure 2: Neural network process data as a black-box.

The MatLab toolbox Neural Network functions *newff()* is used to initialize the architecture of the network, *train()* is used to train the network. Then the MatLab Simulink function *sim()* is used for the neural network prediction.

Activation functions are chosen to process information from input nodes $X_i's$ (i=0,1,...,a) and hidden nodes $Z_j's$ (j=0,1,...,b), where X_0 and Z_0 are the bias (intercept). The typical activation functions are the log-sigmoid (1) and logistic functions (2). These two functions return a monotone increasing probability function in $(0,1)$.

$$\sigma(v) = 1/(1+e^{-v}). \tag{1}$$

$$logit(v) = exp(v)/(1 + exp(v)) \tag{2}$$

The log-sigmoid function is chosen for our analysis, where

$$Z_j = \frac{1}{1 + exp\{(-1)(\alpha_0 + \alpha_j^T X)\}} \tag{3}$$

If there is more than one hidden layer, apply activation functions between layers. For binary classification, the output layer can be viewed as two parts, linear combination and transformation. There is another set of weights applied to the linear combination,

$$Y = \beta_0 + \beta_j^T Z, \quad j = 1, ..., b. \tag{4}$$

A transfer function such as log-sigmoid or logistic function will be applied to the linear combination result, and we will have the probability output

$$Prob = 1/(1 + e^{-Y}). \tag{5}$$

A threshold value is picked for the classification. A complete neural net figure is shown in 1.

$$output = \begin{cases} 0, & \text{if Prob} < \text{threshold value;} \\ 1, & \text{if Prob} \geq \text{threshold value.} \end{cases} \tag{6}$$

To build (or train) the neural network, we need to estimate all of the weight parameters $\alpha_i's$ and $\beta_j's$. For the "$a - b - 1$" neural network, there are $b(a + 1) + (b + 1)$ parameters in total including the intercepts (or noise) need to be

estimated. For classification neural network, mean-squared error is used as a measure of fit,

$$mse = \frac{1}{N}\sum_{i=1}^{N} err(i)^2 \tag{7}$$

The Levenberg-Marquardt algorithm (Neural Network Toolbox User's Guide), an approximate gradient descent procedure (trainlm() in matlab) which gives an approximation to the Hessian in Newton's method, is used to adjust the weight parameters to minimize MSE.

2.2 Bagging and Boosting

It is often possible to increase the accuracy of the classification by averaging the decisions of an ensemble of classifiers. Boosting and bagging are two techniques for such a purpose, and they work better for unstable learning algorithms such as neural networks, logistics regression, and decision trees.

Bagging involves fitting the model, including all potential data points, on the original training set. Bootstrap samples with replacement of the original training set of size up to the size of the training set are generated. Some of the data points can appear more than once while others don't appear at all. By averaging across resamples, bagging effectively removes the instability of the decision rule. Thus, the variance of bagged prediction model is smaller than if we fit only one classifier to the original training set.(Inoue, A. Kilian, L., 2005). Bagging also helps to avoid overfitting.

The idea of boosting is to increase the strength of a a weak learning algorithm. According to a rule of thumb, a weak learning algorithm should be better than random guessing. For a binary classifier, the weak learning hypothesis is getting 50% right. Boosting trains a weak learner a number of times, using a reweighted version of the original training set. Boosting trains the first weak learner with equal weight on all the data points in the training set, then trains all other weak learners based on the the updated weight. The data points wrongly classified by the preview weak learner get heavier weight, and the the correctly classified data points get lighter weight. This way, the next classifier will attempt to fix the errors make by the previous learner.

There are several boosting algorithm, including AdaBoost, AdaBoost.M1, AdaBoost.M2, and AdaBoost.R (Freund, Y. and Schapire,R. 1995). AdaBoost is for binary classification problems, AdaBoost.M1 and .M2 are for multiple classification problems, and AdaBoost.R is for regression. We modify the AdaBoost algorithm so that it does not require the weak learning hypothesis, since sometimes the unstable neural network has error slightly higher than 50%. Instead of applying weights to each data point in the original training set, our modified boosting algorithm bootstraps resample the training set with updated weights.

Consider a training set D with N data points $(x_1, y_1), ..., (x_N, y_N)$, where $y_i s$ are the targets coded as $y \in \{0, 1\}$, and a testing set D_T with N_T data points. A binary classifier neural network G: $G(X) \rightarrow [0, 1]$.

Boosting Algorithm:

1. *Initialize the observation weights* $w_i = 1/N, i = 1, 2, ..., N$..
2. *For m = 1 to M:*

   (a) Train a neural network $G_m(x)$ *to the training data.*

   (b) Compute the error rate:

$$err_m = \frac{1}{N} \sum_{i=1}^{N} I(y_i \neq G_m(x_i)).$$

   where X *is the original training data.*

   (c) Compute $\alpha_m = (1 - err_m)/2$.

   (d) Update the weights:

   $w_i = w_i * (1 - \alpha_m)$, *if* $y_i = G_m(x_i)$; *otherwise* $w_i = w_i * (1 + \alpha_m)$.

   (e) Normalize the weight vector W.

   (f) Re-sample training data with replacement and weights w_i.

Bagging Algorithm:

1. *Build the model: for m = 1 to M:*

   (a) Bootstrap sample D_m *of size N with replacement from the original training set D with equal weight.*

   (b) Train a neural network $G_m(x)$ *to the bootstrap sample* D_m.

Predicting:

For $m = 1toM$*:*

Apply G_m *to the testing set* D_T*.*

Classifier using $I\{\sum_{i=1}^{M} G_i(x_i)/M > threshold_value\} \in class1$.

2.3 Support Vector Machine

SVM (Boser, Guyon and Vapnik, 1992) is a popular technique for classification. We will only discuss the SVM used for our binary classification problems. Given a training set of instance-label pairs (x_i, y_i), $i = 1, ..., N$ where $x_i \in R^a$ and $y \in \{0, 1\}$. The idea of SVM is to transfer the X into some space by a kernel function, where the X can be separated by a hyperplane. However, in practice, it is never completely separate. Therefore, we will need to minimize the error. Margin, the distance to the closest data point from both classes, is used as the measure of fit. Vapnik-Chervonenkis theory provides a probabilistic test error bound that is minimized when the margin is maximized(Hastie, Tibshirani and Friedman, 2001). The software package we use to train and test the SVM in the R library e1071. The procedure includes the following

1. Transform data to the format of the software required.

2. Explore the performance of various kernels and parameters.

3. Test the model.

We settled on a 3^{rd} degree polynomial kernel.

2.4 Logistic Regression

The logistic model as applied to the binary classification problem is

$$\begin{aligned} logit[p] &= log[\frac{p}{1-p}] \\ &= \beta_0 + \beta_1 x_{1,i} + \beta_2 x_{2,i} + ... + \beta_a x_{a,i} \end{aligned} \tag{8}$$

where $i = 1, ..., n$ and $p = Pr(Y_i = 1)$.

The target variable or in the logistic regression case, the response, for our binary classification problem is $\{0, 1\}$. However the logistic regression equation does not predict classes directly nor predicate the probability that an observation belongs to one class or the other directly. The logistic regression predicts the log odds that an observation will be in class 1. The log odds of an event is defined as

$$\begin{aligned} logodds(class1) &= log(\frac{Pr(class1)}{1 - Pr(class1)}) \\ &= log(\frac{Pr(class1)}{Pr(class0)}) \\ &= log(Pr(class1)) - log(Pr(class0)) \end{aligned} \tag{9}$$

3 Financial Time Series Compilation

The financial time series analysis in this paper include three groups totaling eight stocks and indices. The daily stock adjusted closed prices [1] of Ap-

[1] Adjusted close price is the close price adjusted for dividends and splits.

ple Computer Inc. ($AAPL$), Microsoft Corp. ($MSFT$), International Business Machines Corp. (IBM), General Motors Corporation (GM), General Electric Co. (GE), S&P 500 INDEX,RTH ($GSPC$), NASDAQ-100 (DRM) (NDX), and DOW JONES COMPOSITE INDEX (DJA) are obtained from a reference database(Yahoo Finance, 2006).

The working dataset of $AAPL$, $MSFT$, IBM, $GSPC$, NDX, and DJA contains 3272 (13 years) stock trading days beginning on August 3, 1993 and ending on July 14, 2006. The working dataset for GM and GE also contains 3272 (13 years) stock trading days from August 12, 1992, to August 12, 2002. The daily log-return (r) of the stock or index is calculated as

$$r_i = log(1 + \frac{P_i - P_{i-1}}{P_{i-1}}) \tag{10}$$

where P_i is the daily adjusted closed stock price.

After converting to daily log-return, the actual working dataset comprises 3271 data points. The first 2520 (10 years) data points will be the training dataset (in sample), and the last 751 (3years) data points will be the testing dataset (outsample).

A quick statistical summary of the empirical data, the percentage of non-negative daily return for both in-sample and out-sample of all eight stocks and indices are shown in table 1.

Stock/Index	In-Sample (n=2520)	Out-Sample (n=751)
GM	51.3	48.2
GE	53.4	49.3
AAPL	51.5	54.6
MSFT	52.2	50.7
IBM	52.8	49.8
NDX	54.0	52.7
GSPC	52.4	55.7
DJA	52.1	56.2
Average	52.3	52.2

Table 1: The actual percentage of daily positive log-return (price moving up) of the eight studied stocks and indices.

4 EXPERIMENTS AND RESULTS:

The financial market is a complex and dynamic system involving a very high degree of uncertainty. Therefore, predicting financial time series is difficult. In general, the prediction of financial time series can usually be categorized into fundamental analysis and technical analysis(Lin, Khan, and Huang, 2002). Fundamental analysis is based on macroeconomic data, and technical analysis is based on the historical data.

In the first experiment, we use the boosting, bagging, and other methods described earlier to predict the direction movement of all eight chosen stocks and indices. For our second experiment, we discuss the statistical performance of the bagging algorithm. We consider testing the change of accuracies and variances as the number of classifiers increases in the bagging process. The third experiment is to determine the effectiveness of the model in relation to its use in driving stock trading decisions.

4.1 Experiment 1:

Suppose we consider the two directional movement of the financial time series, moving up if the log-return is no less than 0 coded as class 1, and moving down if the log-return is less than 0 coded as class 0. Therefore, predicting the directional movement is a binary classification problem. Feeding the same inputs for boosting and bagging procedure, as well as for SVM and logistic regression, we will be able to compare their performance.

The accuracy percentage is used to measure the performance. If the model predicts up and the stock or index moves up including steady then it is correct, otherwise, if the stock or index moves down, it is taken as wrong. If the model predicts down and the stock or index moves down then it is correct, otherwise, if the stock or index moves up or steady, it is taken as wrong.

The architecture of the classifier neural network in the boosting and bagging is "$10 - 20 - 1$". Each of the input vectors is of length 10 and includes the 5 most recent log-return prices, and the indication function of the log-return prices, for a total of 10 values. The hidden layer has 20 nodes. The prediction is classifier as 1 (or up) if the output is greater or equal to a threshold (for instant 0.50), and classed as 0 (or down) otherwise. For comparison purposes, logistic and SVM using the same data to train the model as neural networks.

The accuracy in percentage (or hit rate) of all 8 stock and index over the 751 days of out-of-sample testing data is presented in table 2. The second and third column indicate the percentage of correct predictions by logistic regression and SVM. The fourth to seventh columns show the percentage of correct predictions by the boosting algorithm with M=1, 10, 20, and 50 neural networks classifiers, where M=1 is just the sole Neural Network prediction. The eighth to eleventh columns show the percentage of correct predictions using the bagging algorithm with M=1, 10, 20, and 50 neural network classifiers.

	Other Algorithms		Boosting				Bagging			
Stock	Logistic	SVM	M=1	10	20	50	M=1	10	20	50
GM	50.5	50.2	48.9	48.6	47.4	49.8	54.2	49.0	51.0	50.9
GE	50.2	45.5	53.6	52.6	50.9	46.2	51.0	54.3	52.7	52.4
AAPL	50.7	49.7	57.1	55.3	59.0	57.4	60.3	62.1	59.8	58.9
MSFT	51.8	51.9	53.8	57.4	58.2	59.8	55.1	55.1	55.0	54.9
IBM	50.9	52.2	56.6	49.0	50.0	53.4	56.5	51.5	53.9	54.2
NDX	51.0	51.7	54.3	53.0	55.4	52.5	66.3	61.1	63.1	67.2
GSPC	49.9	51.0	63.2	57.0	59.6	60.5	61.4	57.8	59.8	61.5
DJA	51.3	52.2	65.8	55.7	55.5	49.7	52.9	55.9	56.3	55.4
Average	50.8	50.6	56.7	53.6	54.5	53.7	57.2	55.9	56.5	56.9

Table 2: Out-of-Sample classification accuracy of movement in percentage with threshold of 0.50.

In the second part of experiment 1, we modify the output layer of the classifier neural network while keeping the same input structures. Instead of working with the final output classes, we consider one step backward and focus on the probability output. We classify the output into three classes:

if the output probability is greater than 0.55, then it will be the class of up; if the output probability is less than 0.45, then it will be the class of down; otherwise, it will be the class of unclear. The accuracy prediction of the movement in percentage and the total predictions made (out of 751 possible) on the out-of-sample data set on the indices of NDX, $GSPC$ and DJA are shown in table 3.

Boosting								
Index	M=1	#Pred.	10	#Pred.	20	#Pred	50	#Pred
NDX	59.1	655	67.2	635	68.1	621	74.7	628
GSPC	63.7	498	61.5	576	60.5	582	60.6	597
DJA	55.6	475	54.4	588	52.4	588	48.6	609
Average	59.4	543	61.2	600	60.5	597	61.5	611
Bagging								
Index	M=1	#Pred.	10	#Pred.	20	#Pred.	50	#Pred.
NDX	57.9	669	69.1	625	67.3	640	67.9	632
GSPC	64.5	510	62.3	553	66.1	543	64.7	533
DJA	54.2	502	58.6	524	56.1	494	55.2	504
Average	57.7	560	63.4	567	63.6	559	63.1	556

Table 3: Out-of-Sample classification accuracy of movement in percentage with threshold of 0.55 and 0.45.

In Table 2, it is clear that the accuracy on the directional success of boosting and bagging are significantly higher than the SVM and logistic regression, except for the GM, GE and DJA using the boosting process with 50 classifiers. SVM and logistic regression are not significantly different from the chance (50%). Both boosting and bagging processes with 50 classifiers have more than 60% directional success on the prediction of GSPC. Bagging achieves 67% right classification on the prediction of NDX.

In Table 3, there is a significant increase of the accuracy prediction on the direction movement on the indices of NDX and GSPC by both boosting and bagging. There is a 55% hitrate on DJA by bagging, but that might not be very significant since the index DJA actual moved up 56.2% of the time as shown in Table 1.

Some other researchers have done similar work on the direction movement on some stocks and indices. The predictions by boosting and bagging are significantly higher than the results reported by others. Lendasse et al. describe an approach to forecasting movements in the Belgian Bel-20 stock market index, with inputs including external factors such as security prices, exchange rates and interest rates. Using a Radial Basis Function neural network, they achieve a directional success of 57.2% on the test data (Lendasse, 2000). O'Connor, N. and Madden, G.M. describes a neural network to predicting stock direction movements using external factors. They report a directional success of 53.7% on the test data of Dow Jones Industrial Average (DJIA, Dec.18,2002-Dec.13,2004) (Connor and Madden, 2006).

4.2 Experiment 2

One of the advantages of boosting and bagging algorithms is that they can increase the accuracy of the prediction while reducing the prediction variance. Experiment 2 is designed to examine such statistical properties for the bagging algorithm. This experiment involves two parts. In part one,

we test the bagging process with a difference number of the base classifier neural networks, and examine the accuracy performance on the testing set of stock indices NDX and $GSPC$. We also calculate the variance of the hit rate for the process with difference number of base classifiers.

For m = 1:M,

for i = 1:100,

Run bagging process with m base classifiers, and obtain the hit rate H(i,m).

Calculate the average of H(m) = mean(H(i,m)), and Stdev(m) = stdev(H(i,m)), where i=1,...,100.

Obtain total M hit rates and M standard deviations.

The plot of the mean and standard deviation on the testing data set for the indices of NDX and $GSPC$ is shown in figure 4. We can see that the average hit rates for both indices are increasing while the prediction standard deviations are decreasing as the number of base classifiers increase. The hit rate tends to be stable when there are more than 10 classifiers, however, the standard deviations are still decreasing.

In the second part of experiment 2, we run similar testing as in the first part, but instead of testing the mean and standard deviation for the entire testing data set, we apply the test on the individual data points in the data set.

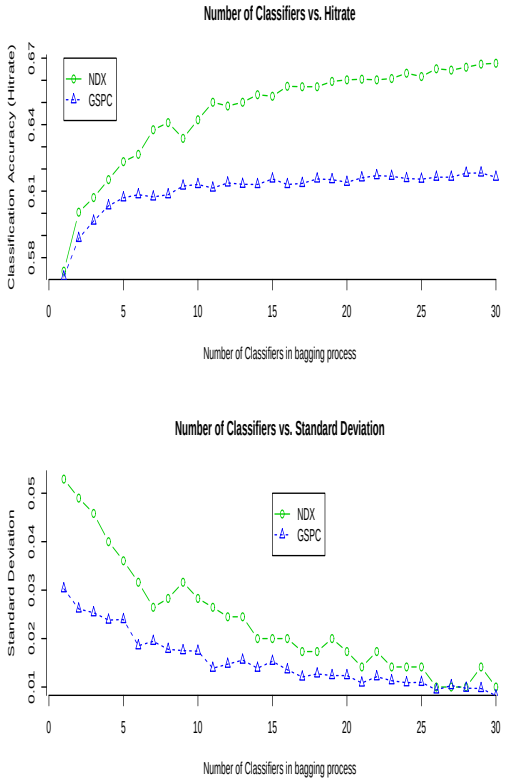

Figure 3: Statistical analysis of the bagging algorithm. Mean and standard deviation of hitrates for the whole out-of-sample data set of NDX and $GSPC$ for M=1,2,...,30.

For m = 1:M,

for i = 1:100,

Run bagging process with m base classifiers, and obtain the hitrate $H_j(i,m)$*, where j=1,...,751.*

Calculate the average of $H_j(m)$ *= mean(*$H_j(i,m)$*), and* $Stdev_j(m)$ *= stdev(*$H_j(i,m)$*).*

Obtain total M hit rates and M standard deviations for each of the 751 *data points in the out-of-sample.*

The plot of the mean and standard deviation of individual data points on the testing data set for the index of $GSPC$ is shown in figure 4. The plot only shows the data points 1, 100, 200, 300, 400, 500, 600 and 700. The average hit rates on most of the data points are increasing and stabilizing as using more base classifiers are used in the bagging process. At the same time, the standard deviations are decreasing. Therefore, the conclusion can be drawn from both parts of experiment 2 that the bagging algorithm decreases the prediction variance without changing the bias.

4.3 Experiment 3

The ultimate need is a measure of the effectiveness of the model in relation to its use in driving decisions to trade stocks. We will use return on investment (ROI) as a measurement to the performance of the models.

We assume that when the market opens we can buy or short sell at yesterday's adjusted closing price. We further assume that the stocks and indices can be traded with fractional amounts. We start with an initial investment of $\$10,000$, and make trading decisions based on the output of the model.

We are testing our strategy on the out-of-sample data including 751 trading days, approximate 3 years. This will be measured as annual ROI (250 trading days, or a calender year). We can add transaction cost to the strategies. While such charges vary between brokerage institutions, we assume

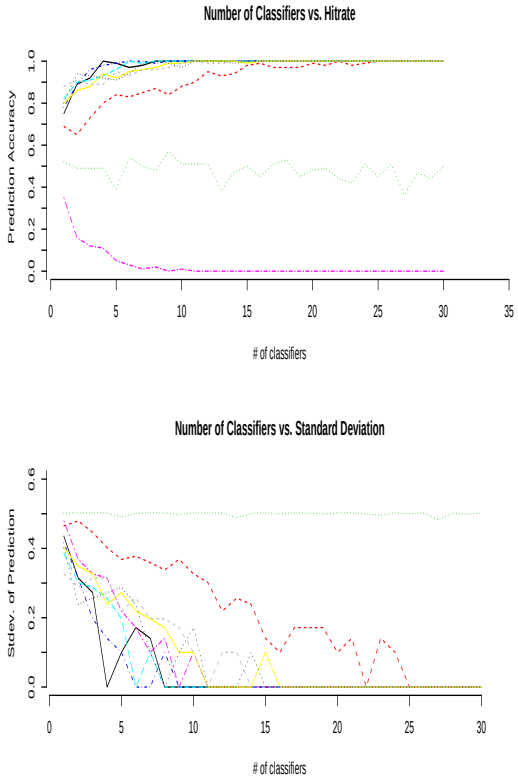

Figure 4: Statistical analysis of the bagging algorithm. Mean and standard of hitrates for the data points 1, 100, 200, 300, 400, 500, 600, and 700 for M=1,2,...,30.

a flat-rate charge of $7 per trade (Scottrade, 2006). All the trading costs are deducted at the end when computing ROI. We are using an initial investment of $\$10,000$ such that the transaction costs would be proportionately less significant.

Strategy 1: Buy and short sell for the model with two classes output of up and down.

1. If the model predicts the price will go up the subsequent day, we will buy the subsequent morning at the price of today's closing price (or opening price), then hold until the model predicts the price will go down some subsequent day, then sell at the closing before that day.
2. If the model predicts the price will go down the subsequent day, we short-sell the subsequent morning at opening price, then hold until the model predicts the price will go down some later day, then buy back at the closing before that day. We will only short sell the amount that is no more than our cumulative investment.

The profits using this strategy 1 and using the output from the part I or the experiment 1 are shown in Table 4. The market growth is defined as, the amount one's investment would have grown if he or she bought on the first day of the period and held until the last day. For instant NDX grew 6.0% annually. The number of trades is the actual trades out of total 751 in the testing period. The annual ROI with and without transaction costs are in percentage.

The ROI without transaction costs is in the third column and the ROI with transaction costs is in the last column of Table 4. They are higher than their corresponding market growth rates except the model output on $MSFT^2$.

Stock/Index	Market	ROI (%)	Avg# of Trade	ROI inc. Tran. Cost (%)
NDX^1	6.0	33.1	124	28.1
NDX^2	6.0	103.6	124	101.4
$GSPC^2$	8.3	26.0	121	20.4
$GSPC^2$	8.3	36.6	121	32.0
$MSFT^1$	-1.1	33.1	133	27.7
$MSFT^2$	-1.1	-0.7	133	-11.2

Table 4: Performance of the model approaches in terms of annual return on investment in percentage. 1 is the boosting with 50 neural networks; and 2 is the bagging with 50 neural networks.

Strategy 2: Modified Buy and short sell for the model with three classes output of up, down and unclear.

1. If the model predicts the price will go up the subsequent day, we will buy the subsequent morning at the price of today's closing price (or subsequent day opening price), then hold until the model predicts the price will go down or no prediction (change of prediction) some subsequent day, then sell at the closing before that day.

2. If the model predicts the price will go down the subsequent day, we short-sell the subsequent morning at opening price, then hold until the model predicts the price will go down or no prediction some later day, then buy back at the closing before that day. We will only short sell an amount that is no more than our cumulative investment.

The profits using this strategy 2 and using the output from the part II or the experiment 1 are shown in Table 5. The boosting and bagging prediction

output on NDX have the ROI with or without traction cost of more than 300% while for the same period of time the market growth rate is 6.0% annually.

Boosting				
Index	Market	ROI (%)	Avg# of Trading	ROI inc. Tran. Cost (%)
NDX	6.0	134.0	102	132.7
GSPC	8.3	17.0	115	10.8
DJA	13.0	-5.9	116	-16.2
Bagging				
Index	Market	ROI (%)	Avg# of Trade	ROI inc. Tran. Cost (%)
NDX	6.0	118.8	110	117.2
GSPC	8.3	5.2	117	-2.8
DJA	13.0	4.4	108	-3.0

Table 5: Performance of the model approaches in terms of return on investment annually in percentage.

5 CONCLUSION AND DISCUSSION:

In this paper, we study the use of boosting and bagging with neural network base classifier to predict financial direction movement. As demonstrated in experiment 1, our bagging and boosting result are superior to other classification methods including SVM and logistic regression in forecasting daily direction movement of all the eight stocks and indices we tested. In the second part of the experiment 2, we were able to obtain 75% prediction accuracy on out-of-sample NDX directional movement. In experiment 2, we can conclude that the bagging process can reduce the prediction variance. Using the output of our models and our buy and short

sell trading strategy described in experiment 3, the return on investment is much greater than the market growth.

The model was trained once with the training data set. It was not retained during the testing period. The first possible extension to this work will be to retrain the model periodically (monthly, weekly, or even daily). By including the most recent data, it is likely to increase the performance of the models. As an evidence of this, during the three-year testing period, the percentage of directional success of the first year is higher than the last two.

For practical consideration of the feasibility of implementing our buy and short sell trading strategy in experiment 3, instead of going all in/out, we may consider lowering the threshold of predicting up, and increase the threshold of predicting down; in another words, be more conservative on short sells, and more aggressive on buys. Another consideration for the implementing of the trading strategy is to invest an amount that is proportional to the degree of certainty of our prediction. For further study, we should consider to include the factor of short-term capital gain tax, as the tax rate can be up to 20%.

REFERENCES

1. A.Inoue and L.Kilian. How useful is bagging in forecasting economic time series? A case study of U.S. CPI Inflation. CEPR Discussion Paper (2005).

2. A.Lendasse, E.de Bodt, V.Wertz, M. Verleysen. Non-linear financial time series forecasting application to the Bel 20 Stock Market Index, European Journal of Economic and Social Systems 14(1)(2000).

3. B.Boser, I.Guyon, and V.Vapnik. A training algorithm for optimal margin classifiers. In Proceedings of the Fifth Annual Workshop on Computational Learning Theory. (1992)

4. C.S.Lin, H.A.Khan, and C.C.Huang. Can the Neuro Fuzzy Model Predict Stock Indexes Better than its Rivals? Discussion papers (2002).

5. H.Schwenk and Y.Bengio. Boosting Neural Networks, Neural Computation 12, 1869-1887 (2000).

6. L.Breiman. Bagging predictors. Machine Learning, 24(2):123140 (1994).

7. Neural Network Toolbox User's Guide.
   http://www-ccs.ucsd.edu/matlab/pdf_doc/nnet/nnet.pdf.

8. N.O'Connor and G.M.Madden. A neural network approach to predicting stock exchange. Knowledge Based Systems Journal, 19 (2006).

9. R.E.Schapire. The strength of weak learnability. Machine Learning. 5(2), 197-227 (1990).

10. Scottrade, http://www.scottrade.com/online_broker_comparison/index.asp (2006).

11. T.Hastie, R.Tibshirani and J.Friedman. The elements of statistical learning; data mining, inference, and prediction,Springer-Verlag, New York, NY (2001).

12. Yahoo Finance, Historic Stock Data. $http://finance.yahoo.com$ (2006).

13. Y.Freund and R.E.Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In Proceedings of the Second Annual European Conference on Computational Learning Theory (1995).

Acknowledgment

I would like to express deep gratitude to my supervisor and friend Kenneth Wilder whose guidance and support were crucial for the successful completion of this thesis paper.