Towards an improved Adaboost Algorithmic Method for Computational Financial Analysis

Victor Chang[1], Taiyu Li[2] and Zhiyang Zeng[2]

1. School of Computing and Digital Technologies, Teesside University, Middlesbrough, UK

2. International Business School Suzhou, Xi'an Jiaotong-Liverpool University, Suzhou, China

victorchang.research@gmail.com

taiyu.li@xjtlu.edu.cn

Zhiyang.Zeng16@student.xjtlu.edu.cn

Abstract

Machine learning can process data intelligently, perform learning tasks and predict possible outputs in time series. This paper presents the use of our proposed machine learning algorithm; an Adaptive Boosting (Adaboost) algorithm, in analyzing and forecasting financial nonstationary data, and demonstrating its feasibility in financial trading. The data of future contracts are used in our analysis. The future used to test the Adaboost algorithm is a contract chosen to study future IF1711, which is combined by "HS300 index and Rb", the deformed steel bar future in Chinese stock market. The predicted data is compared with real world data to calculate accuracy and efficiency. The Adaboost algorithm is combined with an Average True Range-Relative Strength Index (ATR-RSI) strategy, so that it can be applied in automatic trading and therefore demonstrate its practical application. We develop three additional algorithms to enable optimization, large sale simulations and comparing both the predicted and actual pricing values. We performed experiments and large scale simulations to justify our work. We have tested the accuracy and validity of our approach to improve its quality. In summary, our analysis and results show that our improved Adaboost algorithms may have useful and practical implications in nonstationary data analysis.

Keywords: Nonstationary data, Nonstationary time series, machine learning, Back-Propagation Algorithm, Adaptive Boosting Algorithm, ATR-RSI strategy.

1 Introduction

Time series analysis has a number of applications, including business management and market potential prediction. Before performing forecasting and risk estimation, historical data is analyzed in order to discover internal patterns [16]. Research into stationary data is somewhat mature, thus there are mathematical tools and models used to analyze patterns of a stationary sequence. However, most data from the real world are nonstationary. To address the limitations of non-stationarity research, scientists use the differential algorithm to find a stationary data sequence. By performing differentiation, we can locate patterns of the data on a deeper level [4]. But the process of differentiation may sacrifice the quality of analysis [31]. When the order of difference is higher, the analysis and forecasting is less precise. In this paper, we will use a Machine Learning method to analyze and predict nonstationary financial sequences. Machine Learning (ML) involves interdisciplinary research and includes probability theory, statistics, approximation theory, algorithm complexity theory and convex analysis. It is central to artificial intelligence and is the most innovative way to solve practical problems in econometrics and forecasting with non-stationary data analysis. The financial sequence in forecasting task can be performed as a classification task. The classification algorithm assigns a class to a future contract depending on the historical data of the futures. The comparison task will be reached and enhanced by machine learning. Adaptive Boosting Algorithm (Adaboost) is used in machine learning process and will be coupled to ATR-RSI strategy to test the forecasting results. The accuracy rate and the rate of return in the simulation will be compared.

The motivation is as follows. We propose a new algorithm which is fast, efficient and easy to use in a number of contexts. Investors and researchers can use our algorithm to analyze their work and gain foundational understanding about our algorithm. In finance, new and better machine-learning based algorithms are required to make investment and research more competitive and up-to-date. In this paper, we aim to design a model which depends on the provided nonstationary financial data and predicts the future price using machine learning.

Our research contributions are as follows:

- We develop an improved version of Adaboost Algorithms to aim for a stable performance and predictive modeling.
- We improve accuracy and efficiency of financial analysis supported by our performance evaluation.
- We can measure systematic risk with consistency and good performance evaluation.

The general strategy is as follows: Firstly, we collect nonstationary data; the futures of HS300 and Rb. Secondly, we use eighty percent of this data to train our Adaptive Boosting Algorithm and use twenty percent to test the auto-trading strategy called ATR- RSI. In this process, the Adaptive Boosting Algorithm will calculate 7 average prices of different time segments of historical data as 7 features of the nonstationary data. Finally, we analyze the results shown by the model. We use the same trading strategy ATR-RSI on both data predicted by Adaboost and real data to determine the winning rate. The results show that without different process for nonstationary data, Adaboost can predict financial sequences in a reasonable range.

2 Background

This section describes literature and related works for Adaboost and non-Adaboost methods for nonstationary analysis. The understanding of the non-stationarity begins with the stationary process. In order to analyze financial markets and study the Brownian movement, people began to study stochastic processes in the late 19th century. In 1900, Louis Bachelier's doctoral thesis "Speculation Theory" proposed a stochastic analysis of the stock and options markets [2]. With the Brownian movement, the price model can be described using advanced mathematical tools. However, the Brownian motion is a Wiener process; meaning that features such as zero expectation do not fit the price process in the real financial world. In this case, some modified models like the Brownian motion with drift are created. The most well-known is the geometric Brownian motion (GBM), which has been used to mimic stock prices in the Black-Scholes pricing model. According to Hull, the expectation of geometric Brownian motion is independent of the price of the stochastic process (stock price), which is consistent with our expectations of the real market [20]. Hull also stated that the geometric Brownian motion process presents the same "roughness" as the price process we observe in the stock market [20]. In B-S model, there is a strict assumption that the yield (drift rate) and the volatility of asset price are constant over time. The GBM under such an assumption is easy to calculate so it has become the most popular model in mathematical finance.

In classical econometrics, there are two important assumptions for time series: stationarity and constant volatility. The Autoregressive conditional heteroscedasticity (ARCH) model solves the problem caused by the constant volatility assumption. The ARCH model can accurately simulate the volatility of time series variables. It is widely used in empirical research of financial engineering. Obviously, most time series are nonstationary in practice. Although different time series are non-stationary, combinations might be stationary. Based on this, in dealing with the nonstationary data, a common method is differentiating the time series. The difficulties are in determining the difference order. Once the series is stationary, difference will not proceed. The stationarity can be judged by calculating the autocorrelation and partial correlation parameters. Therefore, the traditional time series process is sophisticated and full of uncertainty.

Predicting financial markets in nonstationary series is a significant theme of research and an intricate work, since the prices in financial markets can move in a random process. There are vast factors due to this stochastic behavior and most of them are sophisticated and hard to be forecasted [19] [17]. However, this model should contain factors can represent the features of human behavior and reaction to the financial activity. The subject that studies and explains the factors of human behaviors and psychology in finance is called behavioral finance. The classical behavioral financial model is DSSW, BSV, DHS, HS and BHS which may not be described detail fully in this paper since these are less related to this research. However, behavioral finance shows that humans sometimes are not rational when they make investments. Therefore, human beings have tried to invent systems to help them make investments without the effect of subjective judgement. Suitable transmutation of historical samples will produce samples of the random movement of price. We collect data and input it to the learning algorithm. IF1711 with HS300 and Rb are chosen for the analysis by the algorithm.

The reason we introduce a method based on the machine learning is to avoid those disadvantages of losing information and getting low-accuracy in differential algorithm. Machine learning stimulates people to learn new skills and knowledge. It is also able to reorganize existing knowledge structures and improve their performance [5]. Its application has covered all branches of artificial intelligence, such as expert systems, automatic reasoning, natural language understanding, pattern recognition, computer vision, intelligent robots and other fields. Machine learning is a way to train systems to model neural networks. The aim is to enable systems to work automatically and intelligently based on a large number of data such as voice and image recognition. Similarly, it can be used in the financial field. For example, historical stock data can be used to train financial systems, in order to forecast the rise and fall of the selected stocks. Machine learning uses sophisticated algorithms and models which can enable computer applications to predict unknown data [18]. The artificial intelligent system in this paper will be constructed with neural networks. Neural networks can provide the basic framework of machine learning and DL (deep learning). It has been investigated over a number of years to forecast trends in financial markets and enable automatic computer trading [19].

Adaboost shows a good performance in forecasting nonstationary data in this experiment. Researchers find that Adaboost had the ability to measure the effect of mutual governance variables. Additionally, Adaboost can be effectively used with random forest to enhance the computational performance [15]. This is the reason that the Adaboost will be used in this experiment in this paper. The suggestions of automatic trading are demonstrated by Creamer, who uses boosting for automated planning and trading system [10]. Adaboost is a method presented in this paper. Creamer establishes an efficient automated trading system and tests different parameters. He finally gets positive results that show excellent predictions on selected S&P 500 stocks.

Some nonstationary literature inspires our research direction and shows possibilities for further improvement. A parallelized NLI method with general-purpose computing on the graphics processing unit (G-NLI) is a powerful tool in nonlinear Interdependence (NLI) analysis. As it has high performance in runtime, it is promising to use it in further financial nonstationary analysis, especially in real-time computing and trading [8]. A hierarchical parallel processing framework over a GPU cluster (H-PARAFAC) has unmatched advantages on multidimensional data computing [7]. Its potential value can offer a more precise financial forecasting method since more variables are used in the form of tensors. More different terms and conditions can be considered and added with low feature-loss. A Bayesian tensor factorization (BTF) model supplies the tensors non- stationary analysis [25]. To efficiently deal with the long-time nonstationary financial sequence, high performance dictionary learning methods will be considered and used in our algorithm de- sign [26] [28]. In the progress of the financial nonstationary computing improvement and practice, the nonstationary data would be pretreated using the Bregman method for better and more precise result in optimization problem [27]. In future, our work will explore the possibility of combination of the parallel factor analysis (PARAFAC) methods and the cloud-centered computing platform to increase the real-time nonstationary computing ability and improve the sustainment for complicated analytics [9] [21].

The next section will describe the theory of artificial neural networks. It will start with single layer neural networks and introduce multilayer neural networks which are the basic part of machine learning.

3 Training Methods for nonstationary data analysis

This section describes the methods and theory of training with nonstationary data. The first algorithm is the back-propagation algorithm. The second algorithm is adaptive boosting. This will be used in the experiment to predict financial data and will be used for automatic trading on HS300. Neural networks have been used for algorithm development for machine learning. These include CNN and RNN, as they can produce iterative calculations and predictions for computational analysis [22] with no exceptions for finance. However, the long short term memory (LSTM) is a neural-network based algorithm that has the improvement over CNN and RNN. In our previous work, we demonstrated the usefulness and effectiveness of using LSTM for analyzing and predicting selected Chinese stocks. As shown in Fig 1, our work uses multiple encoder dimensionality reduction, and then creates output to the LSTM extraction. This is similar to deploying the "ReduceMap" method, the opposite of MapReduce as follows [22]. First, all similar characteristics of the dataset congregate into a single or fewer common characteristics. Second, further process and analyze on particular points or instances. This allows better accuracy to be achieved.

Referring to Figure 1, we have applied this principle to the development of our algorithm and also the workflow of the data process, the latter of which will be shown in Figure 3. By applying this principle, it allows us to carefully process and analyze data. We can therefore make more meaningful predictions.

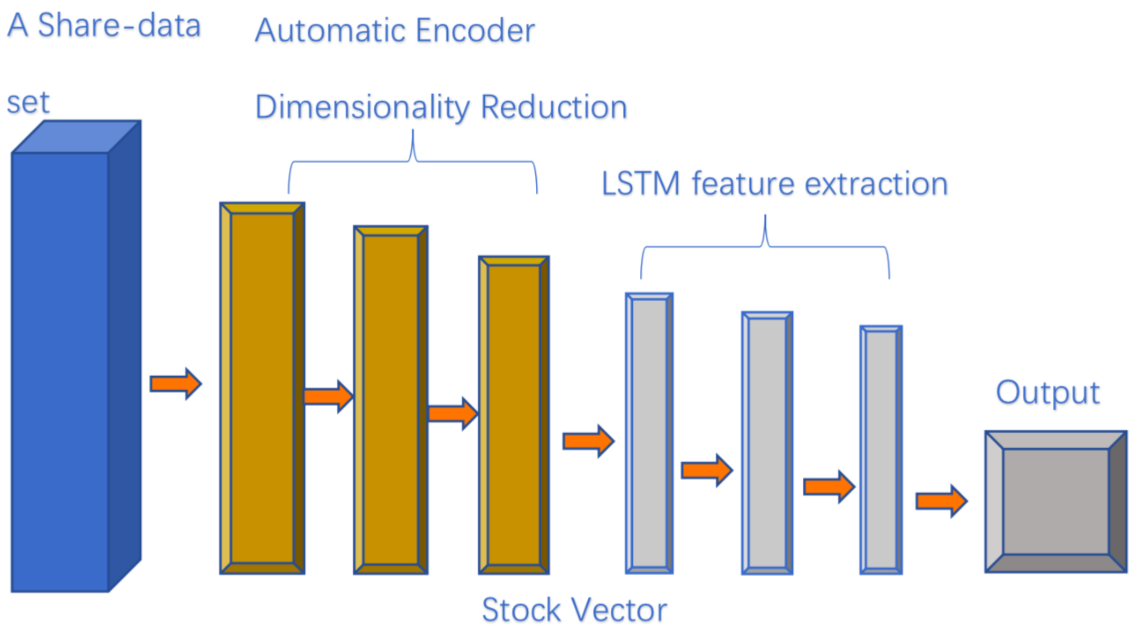


Figure 1: The LSTM

3.1 Back-Propagation Algorithm

Back-Propagation Algorithm (BP) is a way to compute error results from each neuron after a batch of data is processed. In the context of learning, the common method used in back-propagation algorithm to update the weights in neural network theory by computing the gradient of the loss function is gradient descent method. Gradient descent is a first-order repeated optimization algorithm for finding the minimum of a function. In order to use gradient descent to get the local minimum of a function, a step is inversely proportional to the gradient (or approximate gradient) of the function at the present point. In contrast, if a step is relative to the positive proportion of the gradient, the function is close to a local maximum of the function, which is called the gradient ascent. By the way, steepest descent is a special kind of gradient descent. They are similar but differences can still be observed. The Steepest Descent method is used to update the weights [17].

3.1.1 Steepest Descent

The way of steepest descent is extended by Laplace's method for the approximation of an integral.

It can update the weight in the orientation contrary to the gradient vector $\nabla\mu(\omega)$. In this vector, $\mu = \frac{1}{2}e_i^2(n)$ where $e_i(n)$ means the error between output $h_i(n)$ in the network and the desired response $d_i(n)$. It can be shown as $e_i(n) = d_i(n) - h_i(n)$. The following equation is the form of the steepest descent algorithm:

$$w(n) - \eta \, \nabla \mu(w) = w(n + 1).$$

In the equation, η is learning-rate defined before. The correction between time step n and $n + 1$ is

$$\Delta w(n) = w(n + 1) - w(n) = -\eta \, \nabla \mu(w)$$

The equation is useful in approximating $\mu(w(n + 1))$ with the first order Taylor series expansion:

$$\mu(w(n + 1)) \approx \mu(w(n)) + \nabla \mu^T (n) \Delta w(n).$$

Haykin has proven that the rule satisfies the condition of repeated decent [17]. Iterative decent states the next proposition:

Proposition: Beginning at $w(0)$ and then go on a sequence of weight vectors $w(1), w(2), w(3), \cdot \cdot \cdot$

so that the cost function can be decreased at every repeat:

$$\mu(w(n)) > \mu(w(n + 1)),$$

$w(n)$ is the past value of the weight vector and $w(n+1)$ is the value which is updated in the equation.

The input signal which is $h_i(n)$ is collected by neuron i and then produces the output $v_i(n)$ which can be written in the following equation:

$$v_i(n) = \sum_{j=0}^{m} \omega_{ij}(n) h_i(n). \tag{3.1}$$

When $h_0 = 1$, it identifies the bias with weight $w_{i0} = b_i$ in the model. The activation function affords the stimulus from the neuron whose output passing through it. This can be written as:

$$h_i(n) = g_i(v_i(n)).$$

Then making the gradient and using the chain rule to do differential which is

$$\frac{\partial \mu(n)}{\partial \omega_{ij}(n)} = \frac{\partial \mu(n)}{\partial e_{ij}(n)} \frac{\partial e_i(n)}{\partial h_{ij}(n)} \frac{\partial h(n)}{\partial v_i(n)} \frac{\partial v_i(n)}{\partial \omega_{ij}(n)} = -e_i(n) g_i'\big(v_i(n)\big) h_i(n) \tag{3.2}$$

where $e_i(n) = d_i(n) - h_i(n)$ is the derivatives of the error signal, $\mu(n) = \frac{1}{2} e_i^2(n)$ is the error energy, $h_i(n)$ is the function signal from neuron i and $v_i(n)$ is the local field.

3.1.2 The Delta Rule

In machine learning, the delta rule is used to update the weights of inputs. It is a modification $\Delta w_{ij}(n)$ lending itself to $w_{ij}(n)$ and can be expressed as:

$$\Delta w_{ij}(n) = -\eta \frac{\partial \mu(n)}{\partial w_{ij}(n)}. \tag{3.3}$$

In the following equation, η is the parameter of learning as usual and the negative sign results in gradient descent in weight space. It can also be expressed as:

$$\Delta w_{ij}(n) = \eta \delta_i(n) h_j(n) \tag{3.4}$$

where there is the local gradient $\delta_i(n) = e_i(n) g_i^{/}(v_i(n))$. In the delta rule, there are two different cases that should be considered. One case depends neuron i producing an output signal in a hidden layer. The other case is dependent on neuron i producing the signal in the output layer.

Case I: Output Layer

The local gradient can be computed by $\Delta w_{ij}(n) = \eta \delta_i(n) h_j(n)$ where $e_i(n) = d_i(n) - h_i(n)$ when neuron i is in the output layer.

Case II: Hidden Layer

When neuron i is part of a hidden layer, the local gradient can be written as:

$$\delta_i(n) = -\frac{\partial \mu_i(n)}{\partial h_{ij}(n)} \frac{\partial h_i(n)}{\partial v_{ij}(n)} = -\frac{\partial \mu(n)}{\partial h_{ij}(n)} g_i'(v_i(n)). \tag{3.5}$$

Haykin wrote that the cost function $\mu(n) = \frac{1}{2}\sum_{k\in C} e_k(n)^2$ when neuron k is the output neuron. Therefore, the gradient of the cost function is [17]:

$$\frac{\partial \mu(n)}{\partial h_i(n)} = \sum_k e_k \frac{\partial e_k(n)}{\partial h_i(n)}. \tag{3.6}$$

where the error is $e_k(n) = d_k(n) - h_k(n) = d_k(n) - g_k(v_k(n))$ which gives:

$$\frac{\partial e_k(n)}{\partial v_k(n)} = -g_k'(v_k(n)). \tag{3.7}$$

In the end, the gradient of the cost function is:

$$\frac{\partial \mu(n)}{\partial h_i(n)} = -\sum_k e_k(n) g_k'(v_k(n)) w_{ki}(n) = -\frac{\partial \mu(n)}{\partial h_i(n)} = \sum_k \delta_k \omega_{ki}(n) \tag{3.8}$$

where $\frac{\partial v(n)}{\partial h_i(n)} = \omega_{ki}(n)$ and $v_k = \sum_k \omega_k h_i(n)$. So the local gradient for hidden neuron i is

$$\delta_i(n) = g_i'(v_i(n)) \sum_k \delta_k \omega_{ki}(n) \tag{3.9}$$

In summary, the following is the rule for updating the parameters in the network:

1. Using the delta rule to update the parameters:

$$\Delta w_{ij}(n) = \eta \delta_i(n) h_j(n)$$

2. Using the way of case I when neuron i is an output node.
3. Using the way of case II when neuron i is a hidden node and δ_S is needed from the following hidden or output layer in the hidden layer before.

The back propagation algorithm works by going through the data in a backward phase and a forward phase to update the weights. In the following section, the backward and forward phases will be explained.

3.1.3 Backward phase and Forward Phase

Forward Phase

The input data goes through the synaptic weight between layers and does not stop until it finally leaves in the output neurons. The signal satisfies the function:

$$h_i(n) = g\left(\sum_{j=0}^{m=0} \omega_{ij}(n) h_j(n)\right). \tag{3.10}$$

In the equation, g is the activation function, m is the amount in total of inputs except bias, $\omega_{ji}(n)$ is the synaptic weight which makes connection between neuron j and neuron i and $h_j(n)$ is the signal of input of neuron i and the output of neuron j simultaneously. When the neuron i exists in the first hidden layer:

$$h_j(n) = x_j(n). \tag{3.11}$$

In the equation, $x_j(n)$ means the jth input data factor. If the neuron i exists in the output layer:

$$h_i(n) = o_i(n) \tag{3.12}$$

In the equation, $o_i(n)$ means the ith the output vector factor which is compared to the situation resulting in the error $e_i(n)$ from the desired response $d_i(n)$.

Backward Phase

The starting address of a backward phase is the output nodes. The total layers are passed through in the network, where δ is gradient recursively for every neuron in each layer. According to the delta rule, the synaptic weights are updated in this way. The δ is found from multiplying the calculation of the error and the first derivative of its activation function in the output layer. The variation of the weights for the total connections oriented to the output layer can be computed according to (3.4) and (3.9) can be used to compute the δs for the layers which arise before the output layer when the δ is acquired for the output layer. The computation is recursive and uses the way of transmitting the changes to all the synaptic weights to be sustained layer by layer.

3.1.4 Computing δ for Known Activation Functions

This section shows the computation of the hyperbolic tangent function given by
$a \cdot b \cosh^{-2}(bv_i(n))$ and δ for the logistic (sigmoid) function. Their derivatives are:

$$g_i'(v_i(n)) = \frac{a \cdot e^{-av_i(n)}}{(1 + e^{-av_i(n)})^2}, \tag{3.13}$$

$$g_i'(v_i(n)) = a \cdot b \cosh^{-2}(bv_i(n)), \tag{3.14}$$

Noting that $h_i(n) = g_i(v_i(n))$ can be used in the formulas of the derivatives. Below are two formulae of δ depending on whether there exist neurons in a output or hidden layer.

1. **The neurons exist in a hidden layer.**

   The expression in the example of a hidden layer is:

$$\delta_i(n)\begin{cases} a \cdot h_i(n)[1-h_i(n)]\sum_k \delta_k(n)\omega_{ki}(n), & sigmoid \\ \left(\frac{b}{a}\right)[a-h_i(n)][a+h_i(n)]\sum_k \delta_k(n)\omega_{ki}(n), & hyperbolic\ tangent \end{cases}$$

2. **The neurons exist in a output layer.**

since $h_i(n) = o_i(n)$ means the signal of function at the output of neuron i. The formula is:

$$\boldsymbol{\delta_i(n)}\begin{cases} \boldsymbol{a[d_i(n)-o_i(n)]o_i(n)[1-o_i(n)]}, & \boldsymbol{sigmoid} \\ \boldsymbol{\left(\frac{b}{a}\right)[d_i(n)-o_i(n)][a-o_i(n)][a+o_i(n)]}, & \boldsymbol{hyperbolic\ tangent} \end{cases}$$

Where $\boldsymbol{d_i(n)}$ is the desired response.

3.1.5 The Way to Choose Learning Rate

A low value chosen as the learning rate can result in a mutual effect in weights space glabrous at the expense of longer learning rate. A high value chosen for a learning rate can result in a large adjustment. This can lead to the network becoming unstable. In order to accelerate the calculations while maintaining the stability of the network, a momentum term would be integrated and the delta rule defined as follows:

$$\Delta w_{ij}(n) = a\Delta w_{ij}(n-1) + \eta\delta_i(n)h_i(n), \tag{3.15}$$

where a is the momentum constant which is a positive invariable. The delta rule is named the generalized delta rule in this form. If index t moves from 0 to the present time n, it can be projected to a time series. The solution of the equation for $\Delta w_{ij}(n)$ can result in

$$\Delta_{ij}(n) = -\eta\sum_t \alpha^{n-t}\delta_i(t)h_i(t).$$

Hence, the earlier equations can be used to express the following equation:

$$\Delta_{ij}(n) = -\eta\sum_t \alpha^{n-t}\frac{\partial\mu(t)}{\partial\omega_{ij}(t)}.$$

3.1.6 Stopping Criteria

Firstly, setting the W ∗ is the minimum, global or local weight vector. Then taking the gradient of the error to find that it appears in regard to W where it is surely 0 for W = W ∗. The convergence criteria for the BP algorithm is as follows [17]:

If the Euclidean norm of a gradient vector gets the enough little gradient threshold, the BP algorithm is thought to be converged.

It is necessary to calculate the gradient. Haykin suggests that $\mu_{av}(W)$ is settled at $W = W^*$ to improve the stated criteria is following [17]:

If the absolute rate of changes in the average squared error of each epoch is adequately small, then the BP algorithm is convergent.
If the rate of changes is between 0 and 1 percent per epoch, it is expected to be small.

Early-Stopping

Firstly, the training data is separated into two sets; the data of validation and estimation. The network trains with the data of estimation while the data of validation is applied to test generalization capacity. Training is stopped regularly and testing the network on subsets of validation data occurs after every training period. The procedure is as follows:

1. Bias of the MLP and the synaptic weights are fixed after a period of training. The network is to run in its forward mode. The error of validation is estimated for every sample in the subset of the set of validation.
2. Training is restarted for another period and the procedure is iterative when validation is completed.

We will mainly use Adaboost, which is better and more efficient than Back-Propagation Algorithm to train and forecast the financial data.

4 Adaptive Boosting for nonstationary data

4.1 Introduction of Adaptive Boosting

Adaptive boosting (Adaboost) is a machine learning method developed by Freund and Schapire in 1995 [14]. It has been shown to be an accurate learning procedure [5][23]. Adaboost combines all weak classifiers together to create a strong classifier. It can be applied in combination with multiple other learning algorithms to improve performance. The outputs of weak learners are coupled with a weighted accumulation that stands for the eventual results of the weighted classifier. Adaboost is adaptive since later weak learners are tuned to favor instances that were mistakenly classified by fore classifiers. Adaboost is impressible to noise outliers and data. In several questions, it may be less prone to over-fitting, than other approaches. The single one Adaboost learner may be not strong enough, whereas as long as the performance of each learner overwhelm a little than random guesswork, the final model can prove to be a formidable learner. The combination of several learners is different with neural networks. When the Adaboost is in training process, it chooses optimal functions to increase the performance of prediction of the model, reduce the dimension, and possibly shorten the execution time since there is no need to calculate irrelevant features. The figure 2 shows how the Adaboost works. The result is the combination of the four weak classifiers and will provide a great classification. "Weak Learner" is any machine learning algorithm that shows slightly better accuracy than random guesswork. For example, considering the binary classification problem that belongs to about 50% of the samples for each category. Random guess in this case will produce about 50% accuracy. Each weak learner will raise the classification score slightly. In general, it is trivial that weak learners are very simple. The job of weak learners is to get a weak hypothesis which is h_t: $X \rightarrow \{-1, 1\}$ adequate for the D_t distribution. Its error then measures the goodness of the weak hypotheses which can be written as:

$$\epsilon = Pr_{i \sim D_t}[h_t(x_i) \neq y_i] = \sum_{i:h_t(x_i) \neq y_i} D_t(i) \tag{4.1}$$

Note that the error is based on the D_t of weak learner training. In fact, a weak learner could be an algorithm that can make use of the weight D_t on examples of training. Instead, a subset of training examples can be essayed from D_t and the (unweighted) resampling samples can be utilized for training weak learners when this is not possible. See Figure 2 for details.

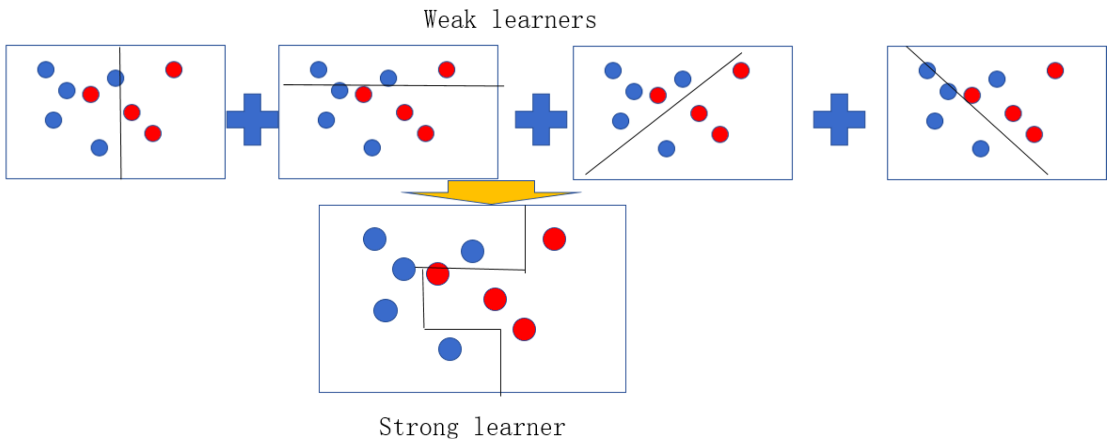


Figure 2: The Adaboost is a strong learner combined by several weak learners

4.2 Adaboost Algorithm

The following algorithm is the Adaptive Boosting Algorithm. Adaboost regulates the errors of the weak hypotheses adaptively. It inputs a set of training$(x_1, y_1), \cdot \cdot \cdot, (x_m, y_m)$ where x_i is in space X and label y_i is in label set Y . Assuming $Y = -1.1$, we use an example to explain the algorithm. If a gambler wants to get maximum value of his/her winnings, he or she makes a decision to design a program which can forecast accurately which horse will win according to the usual data (betting odds for each horse, number of races recently won by each horse, etc.). He or she collects rules of thumb from expert gamblers and use them to his/her advantage. Here the instance x_i can be taken as horse races and the labels y_i give the outcomes of each race.

Algorithm 1 Framework of Adaboost Algorithm for our system.

```
Given $T$ ,suppose $t = 1$, $X_i \in X$, $y_i \in Y = \{-1, +1\}$. The distribution of weak learner satisfies
$D_1(i) = 1/M$.
Define the error as:
    $\epsilon = Pr_{i \sim D_t}[h_{t(x_i) \neq y_i}]$.
where the $h_t$ is the weak hypothesis in $\{-1, 1\}$.
while $t \leq T$ do
    $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
    $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$, where $Z_t$ is a normalization factor.
end while
Output:
    $\mathbf{H(x) = sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$.
```

The weak hypotheses can be seen as the rules of thumb where examined subsets are selected based on the distribution D_t. Each time the weak hypotheses h_t is examined, a parameter α_t is chosen by Adaboost. α_t estimates the importance which is assigned to h_i. Notice that $\alpha_t \geq 0$ if $\epsilon \leq \frac{1}{2}$ and α_t becomes larger with ϵ_t becoming smaller.

The next step is to apply the rule demonstrated in the algorithm 1 above, with the aim to update the distribution D_t. The rule results in an increase in the weight of misclassified samples h_t, and a decreasing weight of properly classified samples.

The final hypotheses H is a weighted vote of plurality of the T weak hypothesis where α_t is the weight portioned to h_t.

Analysis using Adaboost can handle weak hypotheses, such as confidence-rated predictions [24]. It means that the weak hypothesis ht exports a forecasting ht(x) ∈ R whose magnitude |ht (x)| offers a measure of "confidence" in the prediction and whose sign is the label of prediction (-1 or 1) for each instance x. Our algorithm can detect errors, report and correct errors (within the acceptable limit).

Algorithm 2 Identify the types of data and adjust strategies

Given T ,suppose $t = 1$, $X_i \in X$, $y_i \in Y = \{-1, +1\}$. The distribution of weak learner satisfies

```
$D_1(i) = 1/M.$
Define the error as:
        $\epsilon = Pr_{i\sim D_t}[h_{t(x_i)\neq y_i}].$
where the $h_t$ is the weak hypothesis in $\{-1, 1\}$.
while $t \leq T$ do
        $\alpha_t = \frac{1}{2} ln(\frac{1-\epsilon_t}{\epsilon_t})$
    $D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$, where $Z_t$ is a normalization factor.
end while
Output:
        $\mathbf{H(x) = sign}\left(\sum_{t=1}^{T} \alpha_t \boldsymbol{h}_t(x)\right).$
```

4.3 Analysis of training errors

The ability of Adaboost to reduce training errors is its most basic theoretical property. Setting the error E_t of h_t is $\frac{1}{2} - \gamma_t$. Since the hypothesis that expects each class of instance at random has a rate of error of ^{1} (on binary questions), γ_t estimates how much better than random are forecasting of h_t. Schapire and Freund have shown that the error of training of the final hypotheses can be written [14]:

$$\prod_t\left[2\sqrt{\epsilon(1-\epsilon_t)}\right] = \prod_t \sqrt{1-4\gamma_t^2} \leq e^{-2\sum_t \gamma_t^2}. \tag{4.2}$$

Therefore, the training error declines at exponential speed if every weak hypotheses is a little higher than random such that $\gamma_t \geq \gamma$ for some $\gamma > 0$. In practice, it is difficult to understand such boundaries. Despite this, Adaboost is adaptive and adjusts to the error rates of single weak hypotheses.

The bound given in (4.2) which coupled with the bounds on generalization error explained show that Adaboost is equivalent to an optimizing algorithm since it can effectively strengthen and improve a weak learning algorithm several times. By doing so, this enhances the possibilities to become a strong learning algorithm.

4.4 Generalization of errors

Schapire and Freund prove the way to make bounds for the generalization error of the final hypothesis according to its training error, the VC-dimension d which is a normal measure of the "complexity" space of hypotheses for the weak hypothesis space, the sample size m and the number of boosting rounds T [14]. The generalization error is defined as follows with high probability [3]:

$$\hat{P}_r[H(x) \neq y] + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right).$$

In the above equation, $\hat{P}r[\cdot]$ means empiric probability on the training sample. The bounds implies that boosting will result in overfitting when it runs for a high number of rounds or if T becomes too large. However, former research output stated overfitting did not happen during operations over a thousand rounds. Furthermore, Adaboost would run on reducing the error, even when the error of training starts to disappear. This contradicts the spirit of the bound above clearly [5] [13] [23].

In order to research empirical discoveries, Schapire got an alternative approach according to the margins of the examples of training [24]. The definition of margins of the sample (x, y) can be written as:

$$\frac{y \sum_t \alpha_t h_t(x)}{\sum_t \alpha_t}. \tag{4.3}$$

The margin of training sample is between -1 to 1. It becomes positive if and only if H exactly classifies the samples. Furthermore, the dimensions of the margin can be construed as a measurement in the process of forecasting. Schapire found that there are larger margins on the set of training transforming to a superior upper bound on the generalization error [24]. The generalization error is no more than:

$$\hat{P}_r[margin(x,y) \leq \theta] + \tilde{O}\left(\sqrt{\frac{d}{m\theta^2}}\right), \text{ for any } \theta > 0 \text{ with high probability.} \tag{4.4}$$

Notice that the bound does not rely on time entirely. Schapire showed boosting is invasive at decreasing the margin in particular, since it focuses on the samples with the smallest margins [24]. The behaviors of Adaboost can be understood in a game theoretical environment. They are explored by Freund and Schapire [1]. They found that boosting can be seen as an iterative play for some games. Adaboost can be explained to be a particular sample of a more normal algorithm to play iterated games and for approximately finishing a game. This means that boosting has a close relationship with online learning and linear programming.

4.5 Multiclass Classification

There are only binary classification questions in the previous sections and their objects are to distinguish between only two possible classes. In practice, there are more than 2 possible classes of learning questions. They are thus defined as multi-classed. Adaboost can process and simulate this type of sample.

The simplest generalization, named Adaboost.M1 is competent if it is strong enough for the weak learner to acquire reasonable high accuracy [14] on the hard distributions created by Adaboost. Nevertheless, if the weak learner is not able to acquire a rate of more than 50% accuracy operating on these hard distributions, the method is considered as a failure. Some more complicated methods have been developed for the latter case. The algorithm Adaboost.MH designed by Schapire and Singer constructs a set of binary problems for every sample x and possible label y [24]. Freund and Schapire designed the algorithm Adaboost.M2 to replace creating binary questions for every sample x with exact label y and every incorrect label y' [14].

The methods exact extra efforts to design weak learning algorithms. A diverse technique [24], which incorporates method of error-correcting output codes from Dietterich and Bakiri [12], finds similar certifiable bounds to those of Adaboost.MH and M2 are able to be applied with any weak learners to deal with binary labeled data.

4.6 Sophisticated way to process and analyze data

In this section, we describe the experiment that using Adaboost to train data and forecast future financial data. Python 3.6.4 software with *vn.py* was the run-time environment for all the computational analysis. Two packages were required: *numpy* and *talib*. The data was provided by cooperating financial institutions. We uploaded and shared the CSV files to the following link: https://bit.ly/2NeLe6Y. IF1711 data were gathered from 16th April 2010 to 20th March 2017. Rb data were gathered between 3rd June 2014 and 4th May 2017.

The data were divided thus: 80% of the data was applied for training and 20% was applied for testing. It is trivial that all of data are nonstationary. The process of the experiment is shown in the following and the results are presented. The data used in the experiment was from IF1711 and was gathered for the period 16th April 2010 to 20th March 2017. All the data are downloaded from Yahoo Finance. The training data is divided into a training set and a test set which includes 444750 examples. It offers an input matrix of 444750 $\times$ 7 dimension and a vector with 444750 labels. For the prediction experiment, we also use Rb data (deformed steel bar) to compare with IF1711. The Rb data is from 3th June 2014 to 4th May 2017.

The training data was divided as follows:

1. 80% of the data is training data
2. 20% of the data is testing data

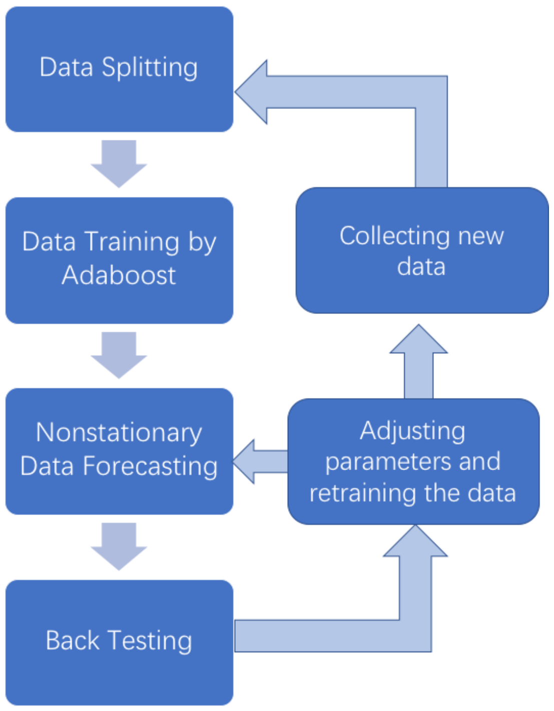

Data Splitting
Data Training by Adaboost
Nonstationary Data Forecasting
Back Testing
Collecting new data
Adjusting parameters and retraining the data

Figure 3: The workflow of the data process

Figure 3 shows the workflow of the data process involved with four steps in a sequential way. If any errors are spotted, we return to the previous step to double check. By following this workflow closely, we can ensure the quality of data and therefore our scientific analysis. In the experiment, we back-tested the HS1711 data from 1st May 2016 to 1st March 2017 in python and used it for automatic trades. We combined Adaboost with ATR-RSI trading strategy. This can be explained as follows.

ATR means average true range. It is a technical analysis volatility indicator originally developed by J. Welles Wilder, Jr. for commodities [29]. Since subtracting the price can calculate the true range and the ATR, the historical volatility is not influenced when the historical price is reversed by subtracting or increasing a constant for each price. Reverse adjustments are often used if short term contracts are stitched together to form a long-term contract. However, the standard procedure used to calculate stock price volatility is not invariant. As a result, analysts and futures traders typically utilize ATR to calculate volatility, whereas stock traders and analysts often use standard deviation of logarithm price ratios. ATR can be calculated thus:

$$T = max[(high - low), /high - close_{prev}/, /low - close_{prev}/]$$

$$A_t = \frac{A_{t-1} \times (n-1) + T_t}{n}$$

$$A = \frac{1}{n}\sum_{i=1}^{n} T_i$$

In this equation, A presents ATR, T is the true range and it extends the range of a day's trading to yesterday's closing price if it was outside of today's range. The main idea of ranges is that they show the commitment or enthusiasm of traders. RSI is the relative strength index, which is a technical indicator applied in analyzing financial markets and was developed by Wilder [29]. It intends to plot the strength in the real world of an asset based on the closing price of the last trading session. Wilder believed that when prices rise quickly, at some point they were thought to be overbought [29]. Similarly, when prices fall very quickly, at several points it is thought they are oversold. In either case, Wilder suggests that it a response or forth coming counter turn [29]. The RSI measures the asset's recent trading ability. The gradient of the RSI is linear relative to the rate at which the trend changes. The moving distance of an RSI is proportional to the size of the movement. Wilder stated that when the RSI exceeds 70 or below 30, the top and bottom will be displayed. Traditionally, an RSI above 70 is considered overbought, and an RSI below 30 is considered oversold. Levels between 30 and 70 are considered neutral and level 50 means no tendency. The calculation can be written as:

$$RS = \frac{SMMA(U, n)}{SMMA(D, n)}$$

$$RSI = 100 - \frac{100}{1 + RS}$$

In the equation, U means an upward change, D means downward change and SMMA is an n-period modified or smoothed moving average which is an exponentially smoothed Moving Average with $\sigma = 1/\text{period}$. If the close which is being higher than the previous close characterizes up periods, then:

$$U = c_{now} - c_{previous}$$

$$D = 0$$

If the close which is lower than the previous close characterizes a down period, then:

$$U = 0$$

$$D = c_{previous} - c_{now}$$

where c means the close. We combine ATR with RSI as the signal of buying or selling in the strategy of automatic trading and then show the results.

4.7 Sophisticated way to process and analyze data

This section aims to describe the algorithm to perform simulation. Optimization is required to simulate large amounts of data. We develop Python-based function to process and simulate. Optimization provides important step to compute large scale simulations quicker. Before this, any errors or unprocessed data, can be processed and analyzed along the way. It is essential to allow processing of data with a better performance. Adam is a stochastic gradient-based optimization method introduced by [21], who mainly use it as a deep learning algorithm. We re-improve it for financial analysis and use it for our optimization, with the explanations as follows. First, data can be loaded for processing and synthesis in a central array in our financial computation called FC. Second, the algorithm check and rank all the data and store them safely in the central array. After saving the data, it is also important to streamline all the data make analysis smoothly. This is enabled by a process called streamline( ). Additionally, the function qualityofservice( ) aims to compute quality of service over a period of time during processing, optimization and refinement. The function optimized means the analysis can be accelerated. The function refine means the analysis can get be further synthesized for a better quality. The process present can be used to enable optimized and refined functions respectively. Both functions are required to ensure analysis is accurate and up-to-date. Experiments will be discussed in Section 5.

Algorithm 3 Performance optimization

```
Load model parameters from database.
Initialize i = 1, MinSmp = 10, FC = [ ]
while(true) do
    j = Acquire input signals
    v = Encode signals j into SDR
    f = Map v into feature space
    s = Extract state identifier for f
```

```
        FC[i] = s                               FC = list of previous states
        check();
        rank( );
        store( );
        streamline( );
        if(i<= MinSmp) then                MinSmp = min. samples for inference
            continue
        else
            p1 = Get state computed index based on FC1[1:i-1]
            p2 = Get state computed index based on FC2[1:i-1]
            g1 = Get refined prediction for the state p1
            g2 = Get refined prediction for the state p2
            optimized = (p1+p2)/2 – f              optimized = optimization achieved
            refined = (g1+g2)/2 – f                refined = refined prediction
            good_noGood = Classify res
            if(good_noGood == True) then
                return NoFault
            else
                return Fault
            end if
        end if
        qualityofservice( );
        present(optimized);
        present(refined);
    end while
```

4.8 Sophisticated way to process and analyze data

We also develop functions and processes to allow large scale simulations, so that we can identify the true performance evaluation. The function "1000simulation" allows 1,000 simulations to be conducted each time, and execution time to be measured during the experiments. The syntax is shorter and cleaner than algorithm 3, since all analysis can be up-to-date and ready for large scale simulations. The function 1000simulation can run 10 times until the end of the statement, in other words, 10,000 simulations. Execution time for each 10,000 simulations can then be recorded. The process can then continue and time to be recorded for the update.

Algorithm 4 Large scale simulations

```
    Load model parameters from database.
    Initialize i = 1, MinSmp = 10, FC = [ ]
    while(true) do
        j = Acquire input signals
```

```
            v = Encode signals j into SDR
            f = Map v into feature space
            s = Extract state identifier for f
            FC[i] = s                              FC = list of previous states
            streamline( );
            optimize( );
            if(i<= MinSmp) then                MinSmp = min. samples for inference
               1000simulation ( );
               time ( );
               i++;
            else
               continue( );
            time( );
            update( );
    end while
```

4.9 Sophisticated way to process and analyze data

We also develop functions and processes to measure the predicted and actual financial values, with 3%, 5% and 10% confidence interval and range of uncertainties allowed. They are represented by functions of “3percent, 5 percent, and 10percent” respectively. We then compare the actual and predicted values between the actual and predicted values with 3%, 5% and 10% differences. We then compare the differences and also measure execution time. We aim to keep the accuracy high with an excellent performance.

Beta is defined the measurement of the system risk and it has been commonly used in financial market to determine the extent of risk. It can be presented by the function beta( ). Values of beta and the execution time can be measured while performing large scale simulations. Their algorithm is as follows.

Algorithm 5 Compare the predicted and actual values and measurement of beta

```
    Load model parameters from database.
    Initialize i = 1, MinSmp = 10, FC = [ ]
    while(true) do
            j = Acquire input signals
            v = Encode signals j into SDR
            f = Map v into feature space
            s = Extract state identifier for f
            FC[i] = s                              FC = list of previous states
            streamline( );
            optimize( );
            if(i<= MinSmp) then                MinSmp = min. samples for inference
```

```
                3percent ( );
                5percent( );
                10percent( );
                beta( );
                time ( );
                i++;
            else
                beta( );
                continue( );
            time( );
            update( );
    end while
```

5 Experiments and Results

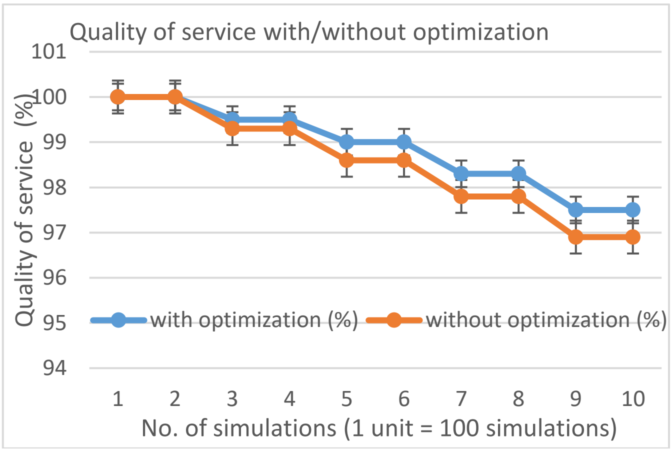
The computer infrastructure setup included the use of a cluster of 100 virtual machines (VMs). Each cluster managed 10 VMs. Each virtual machine had the following hardware platform: Intel(R) Core(TM) i5-2467M CPU 1.6GHz, Intel HD Graphics 3000 384MB, 4 GB 1333 MHz DDR3 memory and 121GB hard disk. During the experiments, the workload could be shared equally to each VM, so that computational analysis could be performed.

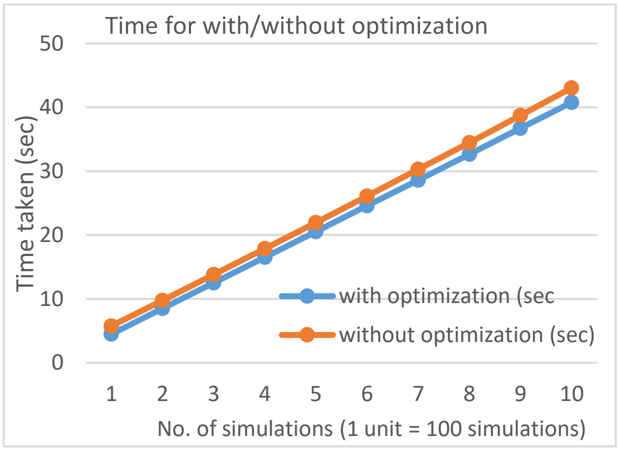
5.1 The Experiments

Our experiment was conducted in the following way. The main idea is similar to Creamer and Freund's use of boosting to predict financial performance [11]. The first process is to input all the data to Adaboost to find the 7 features of the training data. Here we choose 7 features that have different range of moving average of HF1711 since long and short moving averages are able to measure price trends. Since the data is a type of per minute data, we define it as a 5-minute data. This enables convenient calculation. Three major experiments were conducted. First, the performance evaluation with and without optimization over a long period of time were conducted as in Section 4.7. The quality of service, optimized and refined status would have their execution time record. Second, we then performed the large scale simulations, between 1,000 and 10,000, with execution time taken as described in Section 4.8. The aim was to test robustness of our proposed solution. Third, the actual and predicted results were computed and compared. If the difference was within 3%, 5% and 10%, then the outputs would be considered as "accurate". The outputs of our accuracy tests and execution time within 3%, 5% and 10% tolerance were recorded. Fourth, the systematic risk, beta, was also computed for all data analysis. The execution time was also recorded.

5.2 Experiments QoS and execution time with/without optimization

Fig 4 shows the results with/without optimization. The first one is on quality of service (QoS). Results with optimization has better performances. It degrades slightly lower than the one without optimization and keeps within 1-3% better after running 300 simulations. QoS also indicates how our algorithm 1 and 2 can handle with detection and improvement of erroneous data. The second test is on the performance evaluation. The execution time with optimization keeps between 3% and 10% better than the one without optimization. Since the execution time is short, it is not easy to identify the differences until the use of large scale simulations for a better justification. Experimental results show that it is worth for all data to be optimized and get refined status, since it can guarantee a better QoS and a shorter execution time while running more simulations.



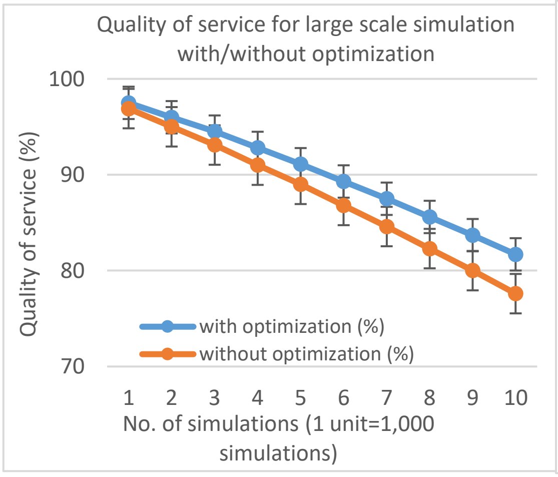


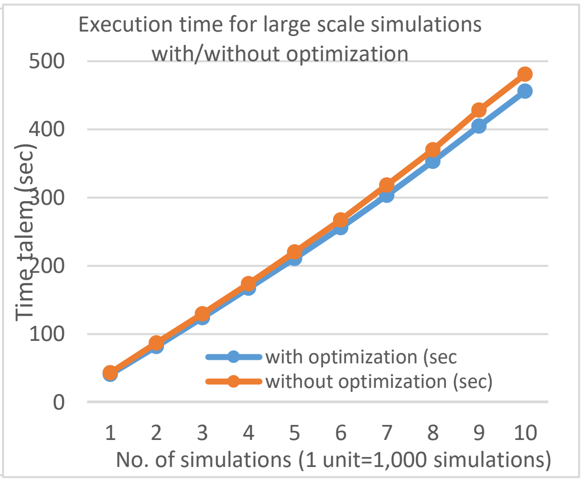
4a: QoS with/without optimization comparison 4b: Execution time with/without optimization

Fig. 4: Experiments of QoS and execution time with/without optimization

5.3 Experiments with large scale simulations

This section shows the experimental results with large scale simulations between 1,000 and 10,000 simulations. The aim was to identify if our proposed work could be resilient enough and allowed the large scale financial analysis to be functioned smoothly. Similarly, QoS and execution time were measured for with and without optimization.





5a: QoS with/without optimization comparison 5b: Execution time with/without optimization

Fig. 5: Experiments of QoS and execution time for large scale simulations with/without optimization

Fig. 5 shows experimntal results of QoS and execution time for large scale simulations with/without optimization. QoS would degrade when the simulations increased. In other words, our algorithms to detect and correct errors would decrease its performance with the increased simulations. Experiments with optimization had up to 10% better QoS than experiments without optimization. For experiments with execution time, the ones with optimization had between 3% and 12% shorter execution time than the ones without optimization. The difference became more noticeable when number of simulations became 70,000 simulations and above.

5.4 Accuracy tests with 3%, 5% and 10% range of uncertainties accepted

We used our algorithms to calculate predicted pricing values, and compared with the actual values. We set the range of uncertainties with 3%, 5% and 10%. In other words, if our predicted values could fall under the upper and lower limits of the actual values, it would then be considered as accurate. If not, then it would not be considered as accurate. They would then be recorded under the percentage of accuracy. The strict tests would be on 3%, since the differences between the predicted and actual pricing values had to be within 3%. Experiments were conducted between 100 and 1,000 simulations since longer execution time would not be suitable for the market trading, and also 1,000 simulations could provide sufficient levels of accuracy. The results were taken based on the average of three records.

Fig. 6 show the experimntal results for accuracy percentages with 3%, 5% and 10% allowed for the range of uncertainties. It was noted when the range was 10%, the predicted and actual pricing values were highly accurate and could stay to 98% when the simulations increased to 1,000. For 3% and 5%, they had similar behaviors. In the 5% difference experiments, accuracy started from 99.5% and declined to 87.1%. In the 3% difference experiments, accuracy started from 99% and declined to 82.1%.

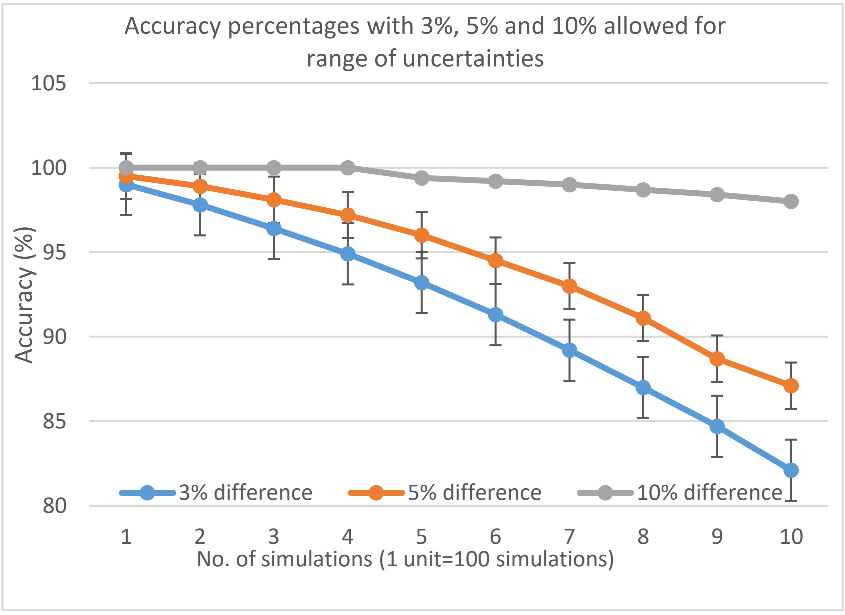


Fig. 6: Accuracy percentages with 3%, 5% and 10% allowed for range of uncertainties.

It is also important to test the accuracy and validity of our approach. This can be achieved by using recall, precision, F-measure, or accuracy, in which we have adopted for the verification process. In the dataset we have, 3%, 5% and 10% differences are used. In other words, the differences between predicted and actual values are within 3%, 5% and 10% respectively.

The TP rate is defined as the percentage of actual values predicted. The FP rate is defined as the percentage of the actual values not predicted, or predicted but outside 10% of range. The FN rate is defined as the percentage not predicted at all. In our case, TN is defined as the percentage of actual values not predicted and computed. But in our case, it is zero percentage which does not happen. The formulae can be written as [6]:

$$Precision = TP/(TP + FP) \text{ and } Recall = TP/(TP + FN);$$

$$Accuracy\ (test) = TP/(TP + FP + FN).$$

A maximum of 10% difference between the actual and predicted value is allowed in our analysis. We use the precision, recall and accuracy (test) to demonstrate the validity and robustness of our approach for these three groups. They are presented in Table 1:

Table 1: Results of the validity and robustness tests.

Accuracy verification	TP rate	FP rate	FN rate	Precision	Recall	Accuracy(test)
3%	0.83	0.10	0.05	0.892	0.943	0.847
5%	0.86	0.10	0.04	0.896	0.956	0.86
10%	0.95	0.03	0.02	0.969	0.979	0.95

Results show that the accuracy rate of analyzing our financial data is 84.7%, 86% and 95% for 3%, 5% and 10% differences between the actual and predicted values respectively. Adaboost can perform well in processing and analyzing our financial data, with low execution time for large scale simulations. This is also known as ATR-RS strategy discussed in the introduction to manage both performance and accuracy to an acceptable level during trading and financial analysis. The 10% difference is considered high with accuracy, however, the challenge is to improve accuracy rate when the differences between the actual and predicted values become smaller.

5.5 The beta test

Beta is considered as the systematic risk in the market and is often measured through regression and large scale data analysis from the financial market. We adopted the same approaches as in the previous sections, and performed large scale simulations. Fig. 7 shows the results for measuring beta between 100 and 1,000 simulations. Values of beta vary between 0.731 and 0.734 regardless of the number of simulations. It means the systematic risk stays close and consistent. Often systematic risk above 0.7 may indicate the market is more volatile than more markets, or the company's recent performance is more volatile. The execution time for performing those experiments was the same as Fig 4(b) with optimization. In other words, all simulations could be completed within 45 seconds.

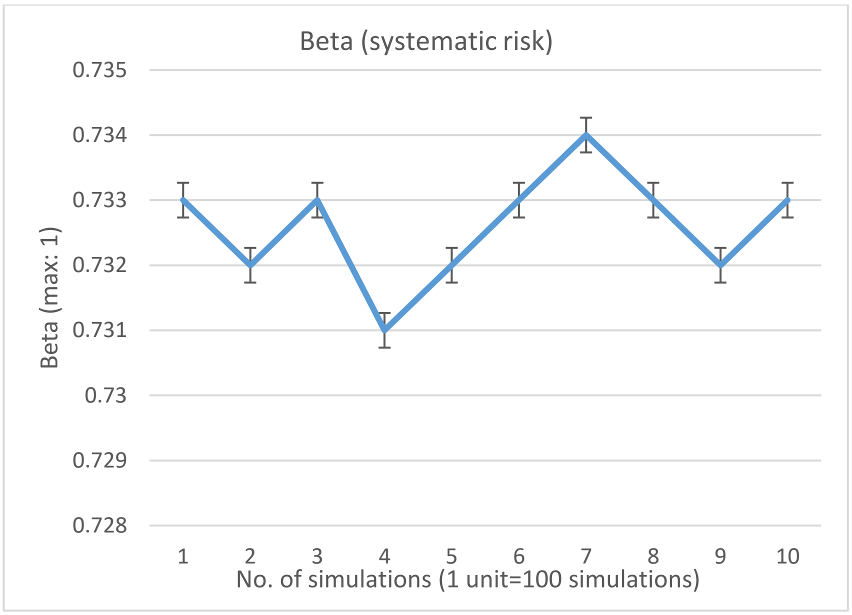


Fig. 7: Values of beta in the large scale simulations

6 Discussion

The methods we developed may benefit the improvement of next-generation computing platform for nonstationary scientific data. This can be considered as a meaningful step for analysis of nonstationary data analysis. Moreover, Adaboost shows positive results in forecasting nonstationary financial data but poor performance on automatic trading withstrategy. Therefore, our research can focus on improving the practical implementations of machine learning on financial trading. We already could achieve better performances with low execution time in large scale simulations, such as 1,000 simulations under 45 seconds with optimization. QoS can also be maintained high, staying above 97% for 1,000 simulations. A challenge is to maintain high QoS for large scale simulations of up to 10,000 simulations. QoS with optimization can stay above 80%. However, this may not be considered well enough for high volume and high speed trading. In the current Adaboost strategy, the most suitable option is to break down financial investment and analysis into smaller sections, and then process and analyze the financial data, and compare the differences between the predicted and actual values. Our accuracy tests show the best outcomes for 10% differences between predicted and actual values, but there are rooms of improvement when the range becomes smaller. Another alternative is to develop pioneering algorithms that can enable accuracy and performance through the big data time series approaches with blind factor approximation [32].

It is important to improve the winning rate of the strategy and allow machine learning to cooperate better with automatic trading. Areas that Adaboost could be applied include firm arbitrage. We will focus on improving the winning rate and will add more real-world parameters such as the knowledge of behavioral finance to add maturity to the model and to continue research of secondary markets.

We have used the live data. The reason for not using k-cross validation technique is as follows. First, speculative movements are likely to happen. We would like to understand the behavioral patterns first by undergoing training data. Upon improvements of our algorithm, we then use the test data to see the extent of our improvement. Second, some Chinese stocks have more unpredictable behaviors. It is useful to divide these stocks into two stages at least; training data and test data, and sometimes in the reversed order, to match predicted outputs and the actual outputs better. Our research direction may use pioneering methods to combine training and testing data and compute both actual and predicted values for two data simultaneously, so that we can improve the quality of our algorithm.

There are two major directions for future research: First, we will improve and perfect our current research in financial area. In particular, we will improve the computational algorithms for nonstationary data incrementally. The newest and most innovative methods such as deep learning will be used to increase the speed, accuracy and robustness of forecasting. Second, we will improve the rates of accuracy. This is an important research area. Our algorithms can be transferrable for other fields. We can apply our existing methods and algorithms to other fields, such as medicine, weather forecasting, and biology in which nonstationary analysis can be used.

7. Conclusion and Future Work

This research focused on machine learning methods to the nonstationary time series. The results showed that we could use our improved Adaboost algorithms to learn and simulate the nonstationary financial sequence well and profit from the process. The main idea of using Adaboost was to divide the training data into several weak classifiers and then combine them as a strong classifier. We tested different nonstationary samples with the same Adaboost and the same strategy. Since the main direction of our research was the futures market, the data we tested was IF1711 which is HS300 and Rb; deformed steel bar. The data was divided as follows: 80% of the data was applied for training and 20% was applied for testing. It was required since a greater presence of training data could help maintain tail of the test error.

In practice, our research outputs could contribute to investment and fund companies to help them manage assets with stable profits and to manage risk. It showed the possibility of using machine learning methods to solve the nonstationary problem. In the process, this might lessen time spent on trading and observing objects. Companies would be able to save costs since they do not need to employ a large number of asset managers. It could also aid low-risk safe investment for investors with low financial knowledge. It could also offer a new perspective of financial data forecasting to help the society gain a more stable financial risk management.

We completed a research study on nonstationary data analysis and applied it for finance. In summary, our paper demonstrated that Adaboost was a suitable method of machine learning, since it was able to make predictions on winning rates, as well as identify points to perform hedging and forecasting. It worked as a predictive method to forecast the future trend of financial markets according to the historical data and trends, and the movement of the current trends. Adaboost results confirmed its excellent performance on predictions, if there was a greater presence of stable and gradual data. Furthermore, it could combine several different decision trees together in a nonrandom way so that it performs better.

This paper demonstrated Adaboost as a machine learning method combined with a trading strategy, with the aim to analyze and forecast a nonstationary financial series. Our proposed work could be applied to different types of classifiers with different trading strategies to fit variety types of markets and investors. Moreover, other than financial area, our proposed work could be used for nonstationary-based simulations and research analysis such weather forecasting with a better performance. We also justified our three major contributions fully. Additionally, Adaboost can also be used for the development of Industry 4.0/5.0, where intelligent algorithms have to process a lot of data with performance, accuracy and efficiency achieved all the times.

References

[1] Auer, P., Cesa-Bianchi, N., Freund, Y., and Schapire, R. E. Gambling in a rigged casino: The adversarial multi-armed bandit problem. In *focs* (1995), IEEE, p. 322.

[2] Bachelier, L. *Théorie de la spéculation*. Gauthier-Villars, 1900.

[3] Baum, E. B., and Haussler, D. What size net gives valid generalization? In *Advances in neural information processing systems* (1989), pp. 81–90.

[4] Box, G. E., and Pierce, D. A. Distribution of residual autocorrelations in autoregressive- integrated moving average time series models. *Journal of the American statistical Association 65*, 332 (1970), 1509–1526.

[5] Breiman, L. Bagging predictors. *Machine learning 24*, 2 (1996), 123–140.

[6] Chang, V., and Ramachandran, M. Towards achieving data security with the cloud computing adoption framework. *IEEE Transactions on Services Computing 9,* 1 (2016), 138– 151.

[7] Chen, D., Hu, Y., Wang, L., Zomaya, A. Y., and Li, X. H-parafac: Hierarchical parallel factor analysis of multidimensional big data. *IEEE Transactions on Parallel and Distributed Systems 28*, 4 (2017), 1091–1104.

[8] Chen, D., Li, X., Cui, D., Wang, L., and Lu, D. Global synchronization measurement of multivariate neural signals with massively parallel nonlinear interdependence analysis. *IEEE Transactions on Neural Systems and Rehabilitation Engineering 22*, 1 (2014), 33–43.

[9] Chen, D., Li, X., Wang, L., Khan, S. U., Wang, J., Zeng, K., and Cai, C. Fast and scalable multi-way analysis of massive neural data. *IEEE Transactions on Computers 64*, 3 (2015), 707–719.

[10] Creamer, G. *Using boosting for automated planning and trading systems*. PhD thesis, Citeseer, 2007.

[11] Creamer, G., and Freund, Y. Learning a board balanced scorecard to improve corporate performance. *Decision Support Systems 49*, 4 (2010), 365–385.

[12] Dietterich, T. G., and Bakiri, G. Solving multiclass learning problems via error- correcting output codes. *Journal of artificial intelligence research 2* (1994), 263–286.

[13] Drucker, H., and Cortes, C. Boosting decision trees. In *Advances in neural information processing systems* (1996), pp. 479–485.

[14] Freund, Y., and Schapire, R. E. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of computer and system sciences 55*, 1 (1997), 119– 139.

[15] Glorot, X., and Bengio, Y. Understanding the difficulty of training deep feedforward neu- ral networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics* (2010), pp. 249–256.

[16] Granger, C. W. Some properties of time series data and their use in econometric model specification. *Journal of econometrics 16*, 1 (1981), 121–130.

[17] Haykin, S. S., Haykin, S. S., Haykin, S. S., and Haykin, S. S. *Neural networks and learning machines*, vol. 3. Pearson Upper Saddle River, 2009.

[18] Hinton, G. E. To recognize shapes, first learn to generate images. *Progress in brain research 165* (2007), 535–547.

[19] Hinton, G. E., Osindero, S., and Teh, Y.-W. A fast learning algorithm for deep belief nets. *Neural computation 18*, 7 (2006), 1527–1554.

[20] Hull, J. C., and Basu, S. *Options, futures, and other derivatives*. Pearson Education India, 2016.

[21] Ke, H., Chen, D., Shah, T., Liu, X., Zhang, X., Zhang, L., and Li, X. Cloud-aided online eeg classification system for brain healthcare: A case study of depression evaluation with a lightweight cnn. *Software: Practice and Experience* (2018).

[22] Pang, X., Zhou, Y., Wang, P., Lin, W., and Chang, V. An innovative neural network approach for stock market prediction. *The Journal of Supercomputing* (2018), 1–21.

[23] Quinlan, J. R. Learning decision tree classifiers. *ACM Computing Surveys (CSUR) 28*, 1 (1996), 71–72.

[24] Schapire, R. E., and Singer, Y. Improved boosting algorithms using confidence-rated predictions. *Machine learning 37*, 3 (1999), 297–336.

[25] Tang, Y., Chen, D., Wang, L., Zomaya, A. Y., Chen, J., and Liu, H. Bayesian tensor factorization for multi-way analysis of multi-dimensional eeg. *Neurocomputing 318* (2018), 162–174.

[26] Wang, L., Liu, P., Song, W., and Choo, K.-K. R. Duk-svd: dynamic dictionary updating for sparse representation of a long-time remote sensing image sequence. *Soft Computing 22*, 10 (2018), 3331–3342.

[27] Wei, J., Huang, Y., Lu, K., and Wang, L. Fields of experts based multichannel com- pressed sensing. *Journal of Signal Processing Systems 86*, 2-3 (2017), 111–121.

[28] Wei, J., Wang, L., Liu, P., and Song, W. Spatiotemporal fusion of remote sensing images with structural sparsity and semi-coupled dictionary learning. *Remote Sensing 9*, 1 (2016), 21.

[29] Wilder, J. W. *New concepts in technical trading systems*. Trend Research, 1978.

[30] Yan, D., Zhou, G., Zhao, X., Tian, Y., and Yang, F. Predicting stock using microblog moods. *China Communications 13*, 8 (2016),244–257.

[31] Yao, X., Wang, X.-d., Zhang, Y.-X., and Quan, W. Summary of feature selection algorithms. *Control and Decision 27*, 2 (2012), 161–166.

[32] Chen, D., Tang, Y., Zhang, H., Wang, L., and Li, X. Incremental Factorization of Big Time Series Data with Blind Factor Approximation", IEEE Transactions on Knowledge and Data Engineering, (99):1-18, DOI: 10.1109/TKDE.2019.2931687, 2019.