Stock market prediction using Multi Expression Programming

Crina Groşan, Ajith Abraham, Sang Yong Han and Vitorino Ramos

Department of Computer Science
Babeş-Bolyai University, Kogălniceanu 1
Cluj-Napoca, 3400, Romania.
School of Computer Science and Engineering
Chung - Ang University
221, Heukseok-Dong, Dongjak-Gu
Seoul 156-756, Korea
CVRM-GeoSystems Centre
Technical University of Lisbon, Portugal
cgrosan@cs.ubbcluj.ro, ajith.abraham@ieee.org, hansy@cau.ac.kr,
vitorino.ramos@alfa.ist.utl.pt

Abstract. The use of intelligent systems for stock market predictions has been widely established. In this paper we introduce a genetic programming technique (called Multi-Expression programming) for the prediction of two stock indices. The performance is then compared with an artificial neural network trained using Levenberg-Marquardt algorithm, support vector machine, Takagi-Sugeno neuro-fuzzy model, a difference boosting neural network. We considered Nasdaq-100 index of Nasdaq Stock MarketSM and the S&P CNX NIFTY stock index as test data.

1 Introduction

Prediction of stocks is generally believed to be a very difficult task. The process behaves more like a random walk process and time varying [18],[6]. The obvious complexity of the problem paves way for the importance of intelligent prediction paradigms [20], [7]. During the last decade, stocks and futures traders have come to rely upon various types of intelligent systems to make trading decisions [1], [2],[5],[15],[11] . In this paper we perform a comparison of five different intelligent paradigms. Two well-known stock indices namely Nasdaq-100 index of Nasdaq^{SM} [9] and the S&P CNX NIFTY stock index [10] are used in experiments. Nasdaq-100 index reflects Nasdaq's largest companies across major industry groups, including computer hardware and software, telecommunications, retail/wholesale trade and biotechnology. The Nasdaq-100 index is a modified capitalization-weighted index, which is designed to limit domination of the index by a few large stocks while generally retaining the capitalization ranking of companies. Similarly, S&P CNX NIFTY is a well-diversified 50 stock index accounting for 25 sectors of the economy [10]. It is used for a variety of purposes such as benchmarking fund portfolios, index based derivatives and index funds.

The CNX Indices are computed using market capitalization weighted method, wherein the level of the Index reflects the total market value of all the stocks in the index relative to a particular base period.

Our research is to investigate the behavior of MEP technique for modeling the Nasdaq-100 and NIFTY stock market indices so as to optimize the performance indices (different error measures, correlation coefficient and so on). The results obtained by MEP are compared to the results obtained by other four techniques. The four techniques used in experiments are: an artificial neural network trained using the Levenberg-Marquardt algorithm, support vector machine [17], difference boosting neural network [14], a Takagi-Sugeno fuzzy inference system learned using a neural network algorithm (neuro-fuzzy model) [8].

We analyzed the Nasdaq-100 index value from 11 January 1995 to 11 January 2002 and the NIFTY index from 01 January 1998 to 03 December 2001. For both the indices, we divided the entire data into almost two equal parts. In Section 2, we briefly describe the MEP technique used in the experiments. In section 3 a short description of the others techniques used in this paper namely artificial neural network (ANN), support vector machines (SVM) [17], Neuro-Fuzzy System and Difference Boosting Neural Network (DBNN) is presented followed by experimentation setup and results in Section 4. Some conclusions are also provided towards the end.

2 Multi Expression Programming

Multi Expression Programming (MEP) technique description and features are presented in this section.

2.1 Solution Representation

MEP genes are (represented by) substrings of a variable length. The number of genes per chromosome is constant. This number defines the length of the chromosome. Each gene encodes a terminal or a function symbol. A gene that encodes a function includes pointers towards the function arguments. Function arguments always have indices of lower values than the position of the function itself in the chromosome.

The proposed representation ensures that no cycle arises while the chromosome is decoded (phenotypically transcripted). According to the proposed representation scheme, the first symbol of the chromosome must be a terminal symbol. In this way, only syntactically correct programs (MEP individuals) are obtained.

An example of chromosome using the sets F= {+, *} and T= {a, b, c, d} is given below:

1: a
2: b
3: + 1, 2
4: c

5: d
6: + 4, 5
7: * 3, 6

The maximum number of symbols in MEP chromosome is given by the formula:

Number_of_Symbols = (n+1) * (*Number_of_Genes* – 1) + 1,
where n is the number of arguments of the function with the greatest number of arguments.

The maximum number of effective symbols is achieved when each gene (excepting the first one) encodes a function symbol with the highest number of arguments. The minimum number of effective symbols is equal to the number of genes and it is achieved when all genes encode terminal symbols only.

The translation of a MEP chromosome into a computer program represents the phenotypic transcription of the MEP chromosomes. Phenotypic translation is obtained by parsing the chromosome top-down. A terminal symbol specifies a simple expression. A function symbol specifies a complex expression obtained by connecting the operands specified by the argument positions with the current function symbol.

For instance, genes 1, 2, 4 and 5 in the previous example encode simple expressions formed by a single terminal symbol. These expressions are:

$E_1 = a$,
$E_2 = b$,
$E_4 = c$,
$E_5 = d$,

Gene 3 indicates the operation + on the operands located at positions 1 and 2 of the chromosome. Therefore gene 3 encodes the expression: $E_3 = a + b$. Gene 6 indicates the operation + on the operands located at positions 4 and 5. Therefore gene 6 encodes the expression: $E_6 = c + d$. Gene 7 indicates the operation * on the operands located at position 3 and 6. Therefore gene 7 encodes the expression: $E_7 = (a + b)$ * $(c + d)$. E_7 is the expression encoded by the whole chromosome.

There is neither practical nor theoretical evidence that one of these expressions is better than the others. This is why each MEP chromosome is allowed to encode a number of expressions equal to the chromosome length (number of genes). The chromosome described above encodes the following expressions:

$E_1 = a$,
$E_2 = b$,
$E_3 = a + b$,
$E_4 = c$,
$E_5 = d$,
$E_6 = c + d$,
$E_7 = (a + b)$ * $(c + d)$.

The value of these expressions may be computed by reading the chromosome top down. Partial results are computed by dynamic programming and are stored in a conventional manner.

Due to its multi expression representation, each MEP chromosome may be viewed as a forest of trees rather than as a single tree, which is the case of Genetic Programming.

2.2 Fitness assignment

As MEP chromosome encodes more than one problem solution, it is interesting to see how the fitness is assigned.

The chromosome fitness is usually defined as the fitness of the best expression encoded by that chromosome.

For instance, if we want to solve symbolic regression problems, the fitness of each sub-expression E_i may be computed using the formula:

$$f(E_i) = \sum_{k=1}^{n} |o_{k,i} - w_k|,$$

where $o_{k,i}$ is the result obtained by the expression E_i for the fitness case k and w_k is the targeted result for the fitness case k. In this case the fitness needs to be minimized.

The fitness of an individual is set to be equal to the lowest fitness of the expressions encoded in the chromosome:

When we have to deal with other problems, we compute the fitness of each sub-expression encoded in the MEP chromosome. Thus, the fitness of the entire individual is supplied by the fitness of the best expression encoded in that chromosome.

2.3 MEP strengths

A GP chromosome generally encodes a single expression (computer program). By contrast, a MEP chromosome encodes several expressions. The best of the encoded solution is chosen to represent the chromosome (by supplying the fitness of the individual).

The MEP chromosome has some advantages over the single-expression chromosome especially when the complexity of the target expression is not known. This feature also acts as a provider of variable-length expressions. Other techniques (such as Gramatical Evolution (GE) [16] or Linear Genetic Programming (LGP) [4]) employ special genetic operators (which insert or remove chromosome parts) to achieve such a complex functionality.

3 Algorithms used in experiments

3.1 Artificial Neural Networks

The artificial neural network (ANN) methodology enables us to design useful nonlinear systems accepting large numbers of inputs, with the design based solely

on instances of input-output relationships. When the performance function has the form of a sum of squares, then the Hessian matrix can be approximated to $H = J^T J$; and the gradient can be computed as $g = J^T e$, where J is the Jacobian matrix, which contains first derivatives of the network errors with respect to the weights, and e is a vector of network errors. The Jacobian matrix can be computed through a standard backpropagation technique that is less complex than computing the Hessian matrix. The Levenberg-Marquardt (LM) algorithm uses this approximation to the Hessian matrix in the following Newton-like update:

$$x_{k+1} = x_k - [J^T J + \mu I]^{-1} J^T e \tag{1}$$

When the scalar μ is zero, this is just Newton's method, using the approximate Hessian matrix. When μ is large, this becomes gradient descent with a small step size. As Newton's method is more accurate, μ is decreased after each successful step (reduction in performance function) and is increased only when a tentative step would increase the performance function. By doing this, the performance function will always be reduced in each iteration of the algorithm.

3.2 Support Vector Machines (SVM)

The SVM approach transforms data into a feature space F that usually has a huge dimension. It is interesting to note that SVM generalization depends on the geometrical characteristics of the training data, not on the dimensions of the input space. Training a support vector machine (SVM) leads to a quadratic optimization problem with bound constraints and one linear equality constraint. Vapnik [16] shows how training a SVM for the pattern recognition problem leads to the following quadratic optimization problem

$$\text{Minimize: } W(\alpha) = -\sum_{i=1}^{l} \alpha_i + \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} y_i y_j \alpha_i \alpha_j k(x_i, x_j) \tag{2}$$

$$\text{Subject to } \sum_{i=1}^{l} y_i \alpha_i \qquad \forall i : 0 \leq \alpha_i \leq C \tag{3}$$

Where l is the number of training examples $\alpha$is a vector of $l$variables and each component $\alpha_i$corresponds to a training example (x_i, y_i). The solution of (2) is the vector $\alpha^*$for which (2) is minimized and (3) is fulfilled.

3.3 Neuro-Fuzzy System

Neuro Fuzzy (NF) computing is a popular framework for solving complex problems [3]. If we have knowledge expressed in linguistic rules, we can build a Fuzzy Inference System (FIS), and if we have data, or can learn from a simulation (training) then we can use ANNs. For building a FIS, we have to specify the fuzzy sets, fuzzy operators and the knowledge base. Similarly for constructing an ANN for an application the user needs to specify the architecture and learning algorithm. An analysis reveals that the drawbacks pertaining to these

approaches seem complementary and therefore it is natural to consider building an integrated system combining the concepts. While the learning capability is an advantage from the viewpoint of FIS, the formation of linguistic rule base will be advantage from the viewpoint of ANN. We used the Adaptive Neuro Fuzzy Inference System (ANFIS) implementing a Takagi-Sugeno type FIS 7.

3.4 Difference Boosting Neural Network (DBNN)

DBNN is based on the Bayes principle that assumes the clustering of attribute values while boosting the attribute differences [17]. Boosting is an iterative process by which the network places emphasis on misclassified examples in the training set until it is correctly classified. The method considers the error produced by each example in the training set in turn and updates the connection weights associated to the probability $P(U_m?C_k)$ of each attribute of that example ($U_m$is the attribute value andC_k a particular class in k number of different classes in the dataset). In this process, the probability density of identical attribute values flattens out and the differences get boosted up. Instead of the serial classifiers used in the AdaBoost algorithm, DBNN approach uses the same classifier throughout the training process. An error function is defined for each of the miss classified examples based on it distance from the computed probability of its nearest rival.

4 EXPERIMENT RESULTS

Our goal is to optimize several error measures: Root Mean Squared Error (RMSE), Correlation Coefficient (CC), Maximum Absolute Percentage Error (MAP) and Mean Absolute Percentage Error (MAPE):

$$RMSE = \sqrt{\sum_{i=1}^{N} |P_{actual,i} - P_{predicted,i}|},$$

$$CC = \frac{\sum_{i=1}^{N} P_{predicted,i}}{\sum_{i=1}^{N} P_{actual,i}},$$

$$MAP = \max\left(\frac{|P_{actual,\,i} - P_{predicted,\,i}|}{P_{predicted,\,i}} \times 100\right),$$

$$MAPE = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{|P_{actual,\,i} - P_{predicted,\,i}|}{P_{actual,\,i}}\right] \times 100,$$

where $P_{actual,i}$ is the actual index value on day i, $P_{predicted,i}$ is the forecast value of the index on that day and N = total number of days. The task is to have minimal values of RMSE, MAP and MAPE and a maximum value for CC.

4.1 Numerical comparisons

We considered 7 year's month's stock data for Nasdaq-100 Index and 4 year's for NIFTY index. Our target is to develop efficient forecast models that could predict the index value of the following trade day based on the opening, closing and maximum values of the same on a given day. For the Nasdaq-100index the data sets were represented by the 'opening value', 'low value' and 'high value'. NIFTY index data sets were represented by 'opening value', 'low value', 'high value' and 'closing value'. The assessment of the prediction performance of the different connectionist paradigms were done by quantifying the prediction obtained on an independent data set.

4.2 Parameter settings

We used a feed forward neural network with 4 input nodes and a single hidden layer consisting of 26 neurons. We used tanh-sigmoidal activation function for the hidden neurons. The training using LM algorithm was terminated after 50 epochs and it took about 4 seconds to train each dataset. For the neuro-fuzzy system, we used 3 triangular membership functions for each of the input variable and the 27 *if-then* fuzzy rules were learned for the Nasdaq-100 index and 81 *if-then* fuzzy rules for the NIFTY index. Training was terminated after 12 epochs and it took about 3 seconds to train each dataset. Both SVM (Gaussian kernel with $\gamma = 3$) and DBNN took less than one second to learn the two data sets [2].

MEP parameter settings Since we want to optimize RMSE, CC, MAP and MAPE, MEP aim is to obtain an optimal value for a linear combination of these indices. We also want to find an adequate value for population size and chromosome length. In order to choose an adequate combination we performed several experiments. Parameters used by MEP in these experiments are presented in Table 1.

Table 1. Values of MEP parameter settings

Parameter		**Value**
Population size	Nasdaq	100
	Nifty	50
Number of iteration	Nasdaq	60
	Nifty	100
Chromosome length	Nasdaq	30
	Nifty	40
Crossover Probability		0.9
Functions set		+, - , *, /, sin, cos, sqrt, ln, lg, $\log_2$, min, max, abs

In Table 2, the results obtained by MEP for several combinations of RMSE, CC, MAP and MAPE (objective function) for Nasdaq and Nifty are presented.

Table 2. Performance comparison for Nasdaq and Nifty by considering different objective functions

Function to optimize	**Indices Values**			
	RMSE	**CC**	**MAP**	**MAPE**
Results for Nasdaq				
MAP + MAPE + RMSE + CC	0.033	0.989	361.95	30.97
MAP + MAPE + 100 * (RMSE + CC)	0.0229	0.9993	97.43	18.14
$MAP^{0.1}$ + $MAPE^{0.2}$ + RMSE + CC	0.0358	0.97	3577.55	9.30
(MAP^{01} + $MAPE^{0.2}$ + RMSE) * CC	0.021	0.999	96.39	14.33
(MAP^{01} + $MAPE^{0.2}$ + RMSE) * (1 – CC)	0.021	0.998	97.93	18.11
($MAP^{1.5}$ + $MAPE^{0.2}$) * (CC + RMSE)	0.227	0.999	97.431	18.133
($MAP^{0.05}$ + $MAPE^{0.2}$) + (RMSE + CC)	0.021	0.999	135.53	15.36
Results for Nifty				
MAP + MAPE + RMSE + CC	0.0187	0.999	38.99	4.131
MAP + MAPE + 100 * (RMSE + CC)	0.0161	0.991	42.98	3.78
$MAP^{0.1}$ + $MAPE^{0.2}$ + RMSE + CC	0.0134	0.993	43.71	3.24
(MAP^{01} + $MAPE^{0.2}$ + RMSE) * CC	0.094	0.896	1254.45	27.79
(MAP^{01} + $MAPE^{0.2}$ + RMSE) * (1 – CC)	0.0264	0.98	65.66	6.18
($MAP^{1.5}$ + $MAPE^{0.2}$) * (CC + RMSE)	0.0163	0.997	31.7	3.72
($MAP^{0.05}$ + $MAPE^{0.2}$) * (RMSE + CC)	0.0227	0.993	65.66	5.14

As evident from Table 2, if we use MEP to optimize the objective function as (MAP + MAPE + RMSE + CC) results obtained were not so good mainly due to the great difference between MAP and MAPE (which are greater than 1) and RMSE and CC whose values are between 0 and 1. So, we have to combine these indices by taking into account the differences. Consequently, the objective function is modeled as (MAP + MAPE + 100 * (RMSE + CC). Results indicate this combination is better than the first one. Further several other combinations were modeled mainly by considering MAP and MAPE at powers smaller than 1. It is found that some of the results obtained are better than those obtained by the previous combinations. The best result for Nasdaq is obtained by considering fourth combination (Table 2).

It is interesting to note that the best combination obtained for Nasdaq seems to give the worst results to model the Nifty index. Experiments indicate the best combination is given by ($MAP^{1.5}$ + $MAPE^{0.2}$) * (CC + RMSE). MEP also obtains a good result for Nasdaq using the same combination but not the best one.

Next two experiments analyze the results obtained by MEP by considering different population sizes and different values for chromosome length. The values for the other parameters are those from Table 1. Population size was considered

150 for both test data. Results obtained in 10 different runs were averaged. Also the best results obtained are presented.

Results obtained for different values for population size are presented in Table 3 for both Nasdaq and Nifty test data and results obtained for variable chromosome length are presented in Table 4.

Table 3. MEP performance comparison for Nasdaq and Nifty by considering different values for population sizes.

			Population size			
			50	**100**	**150**	**200**
Nasdaq	**RMSE**	Best	0.022	0.032	0.035	0.0168
		Average	0.024	0.027	0.03	0.022
	CC	Best	0.999	0.097	0.97	0.992
		Average	0.995	0.984	0.98	0.997
	MAP	Best	97.43	103.31	103.31	103.31
		Average	109.59	100.37	109.33	100.7
	MAPE	Best	18.13	9.02	9.08	8.69
		Average	23.32	13.58	18.8	18.23
Nifty	**RMSE**	Best	0.0187	0.163	0.015	0.0138
		Average	0.02	0.0196	0.0197	0.019
	CC	Best	0.999	0.997	0.999	0.999
		Average	0.991	0.979	0.978	0.988
	MAP	Best	38.99	31.7	27.64	30.03
		Average	53.02	42.225	41.85	48.81
	MAPE	Best	4.131	3.72	3.17	3.027
		Average	4.9	4.66	4.81	4.34

As depicted in Table 3, best results for Nasdaq is obtained using a population of 100 individuals and for Nifty using a population of 150 individuals.

As we can see from Table 4 the best results for both Nasdaq and Nifty is obtained by using a chromosome length of 40.

Results analysis and discussions Table 6 summarizes the results achieved for the two stock indices using the five intelligent paradigms (SVM, NF, ANN, DBNN, MEP). Greater values for CC and lower values for RMSE, MAP and MAPE indicate a better result.

5 CONCLUSIONS

In this paper we apply a new genetic programming technique called Multi Expression Programming (MEP)for modeling stock indices. The performance of MEP (empirical results) when compared to the four intelligent paradigms clearly indicate that it is a novel promising technique for function approximation problems. It is to be noted that MEP technique gave the lowest MAP values for both

Table 4. MEP performance comparison for Nasdaq and Nifty by considering different values for chromosome length.

			Chromosome length			
			20	**30**	**40**	**50**
Nasdaq	**RMSE**	Best	0.021	0.032	0.028	0.0165
		Average	0.021	0.027	0.024	0.022
	CC	Best	0.998	0.976	0.977	0.993
		Average	0.998	0.987	0.985	0.994
	MAP	Best	97.43	103.31	103.31	103.31
		Average	97.43	100.38	118.5	115.55
	MAPE	Best	18.11	9.02	8.91	8.53
		Average	18.12	13.52	18.74	15.86
Nifty	**RMSE**	Best	0.0187	0.0169	0.015	0.014
		Average	0.0193	0.023	0.0197	0.02
	CC	Best	0.999	0.990	0.999	0.992
		Average	0.994	0.977	0.98	0.981
	MAP	Best	38.99	42.98	27.64	34.87
		Average	43.37	52.1	38.78	40.67
	MAPE	Best	4.125	4.08	3.17	3.30
		Average	4.33	5.68	4.81	4.75

Table 5. Results obtained by intelligent paradigms for Nasdaq and Nifty test data

	SVM	**NF**	**ANN**	**DBNN**	**MEP**
Test results for Nasdaq					
RMSE	0.0180	0.0183	0.0284	0.0286	0.021
CC	0.9977	0.9976	0.9955	0.9940	0.999
MAP	481.50	520.84	481.71	116.98	96.39
MAPE	7.170	7.615	9.032	9.429	14.33
Test results for Nifty					
RMSE	0.0149	0.0127	0.0122	0.0225	0.0163
CC	0.9968	0.9967	0.9968	0.9890	0.997
MAP	72.53	40.37	73.94	37.99	31.7
MAPE	4.416	3.320	3.353	5.086	3.72

stock indices. The fluctuations in the share market are chaotic in the sense that they heavily depend on the values of their immediate forerunning fluctuations. In this research, our main task was to optimize several error measures namely RMSE, CC, MAP and MAPE.

Taking into account of the No Free Lunch Theorem (NFL) [19], our research using real world stock data also reveals that it is difficult for one of the intelligent paradigms to perform well for different stock indices.

References

1. A. Abraham and A. AuYeung. *Integrating Ensemble of Intelligent Systems for Modeling Stock Indices*, In Proceedings of 7th International Work Conference on Artificial and Natural Neural Networks, Lecture Notes in Computer Science- Volume 2687, Jose Mira and Jose R. Alverez (Eds.), Springer Verlag, Germany, pp. 774-781, 2003.
2. A. Abraham, N. S. Philip and P. Saratchandran. *Modeling Chaotic Behavior of Stock Indices Using Intelligent Paradigms*. International Journal of Neural, Parallel & Scientific Computations, USA, Volume 11, Issue (1&2) pp. 143-160, 2003.
3. A. Abraham. *Neuro-Fuzzy Systems: State-of-the-Art Modeling Techniques, Connectionist Models of Neurons*, Learning Processes and Artificial Intelligence, Springer-Verlag Germany, Jose Mira and Alberto Prieto (Eds.), Granada, Spain, pp. 269-276, 2001.
4. M. Brameier and W. Banzhaf W. *Explicit control of diversity and effective variation distance in Linear Genetic Programming*. In Proceedings of the fourth European Conference on Genetic Programming, Springer-Verlag Berlin, 2001.
5. E.B. Del Brio, A. Miguel and J. Perote. *An investigation of insider trading profits in the Spanish stock market*, The Quarterly Review of Economics and Finance, Volume 42, Issue 1, pp. 73-94, 2002.
6. F.E.H. Tay and L.J. Cao. *Modified Support Vector Machines in Financial Time Series Forecasting*, Neurocomputing 48(1-4): pp. 847-861, 2002.
7. W.Huang , S.Goto and M. Nakamura. *Decision-making for stock trading based on trading probability by considering whole market movement*, European Journal of Operational Research, Volume 157, Issue 1, (16), pp. 227-241, 2004.
8. J.S.R. Jang, C.T. Sun and E. Mizutani. *Neuro-Fuzzy and Soft Computing: A Computational Approach to Learning and Machine Intelligence*, Prentice Hall Inc, USA, 1997.
9. Nasdaq Stock Market^{SM}: http://www.nasdaq.com.
10. National Stock Exchange of India Limited: http://www.nse-india.com.
11. K.J. Oh and K.J. Kim. *Analyzing stock market tick data using piecewise nonlinear model*, Expert Systems with Applications, Volume 22, Issue 3, pp. 249-255, 2002.
12. M. Oltean and C. Grosan. *A Comparison of Several Linear GP Techniques*. Complex Systems, Vol. 14, Nr. 4, pp. 285-313, 2004
13. M. Oltean, C. Grosan. *Evolving Evolutionary Algorithms using Multi Expression Programming*, Proceedings of The 7th European Conference on Artificial Life, Dortmund, Germany, pp. 651-658, 2003.
14. N.S. Philip and K.B. Joseph. *Boosting the Differences: A Fast Bayesian classifier neural network*, Intelligent Data Analysis, Vol. 4, pp. 463-473, IOS Press, 2000.

15. R. Rodrguez, F. Restoy and J.I. Pea. *Can output explain the predictability and volatility of stock returns?*, Journal of International Money and Finance, Volume 21, Issue 2, pp.163-182, 2002.
16. C. Ryan C. J.J. Collins and M. O'Neill 1998. *Gramatical Evolution:Evolving programs for an arbitrary language*. In Proceedings of the first European Workshop on Genetic Programming, Springer-Verlag, Berlin, 1998.
17. V. Vapnik. *The Nature of Statistical Learning Theory*, Springer-Verlag, New York, 1995.
18. W.X.Zhou and D.Sornette. *Testing the stability of the 2000 US stock market antibubble*, Physica A: Statistical and Theoretical Physics, Volume 348, (15), pp. 428-452 , 2005
19. D.H. Wolpert and W.G. Macready. *No free lunch theorem for search*, Technical Report SFI-TR-95-02-010. Santa Fe Institute, USA, 1995.
20. J.D. Wichard, C. Merkwirth and M. Ogorzalek. *Detecting correlation in stock market*, Physica A: Statistical Mechanics and its Applications, Volume 344, Issues 1-2, pp. 308-311, 2004