HURST EXPONENT AND FINANCIAL MARKET PREDICTABILITY

Bo Qian **Khaled Rasheed**
Department of Computer Science
University of Georgia
Athens, GA 30601
USA
[qian, khaled]@cs.uga.edu

ABSTRACT
The Hurst exponent (H) is a statistical measure used to classify time series. H=0.5 indicates a random series while H>0.5 indicates a trend reinforcing series. The larger the H value is, the stronger trend. In this paper we investigate the use of the Hurst exponent to classify series of financial data representing different periods of time. Experiments with backpropagation Neural Networks show that series with large Hurst exponent can be predicted more accurately than those series with H value close to 0.50. Thus Hurst exponent provides a measure for predictability.

KEY WORDS
Hurst exponent, time series analysis, neural networks, Monte Carlo simulation, forecasting

1. Introduction

The Hurst exponent, proposed by H. E. Hurst [1] for use in fractal analysis [2],[3], has been applied to many research fields. It has recently become popular in the finance community [4],[5],[6] largely due to Peters' work [7],[8]. The Hurst exponent provides a measure for long-term memory and fractality of a time series. Since it is robust with few assumptions about underlying system, it has broad applicability for time series analysis. The values of the Hurst exponent range between 0 and 1. Based on the Hurst exponent value H, a time series can be classified into three categories. (1) H=0.5 indicates a random series. (2) $0<H<0.5$ indicates an anti-persistent series. (3) $0.5<H<1$ indicates a persistent series. An anti-persistent series has a characteristic of "mean-reverting", which means an up value is more likely followed by a down value, and vice versa. The strength of "mean-reverting" increases as H approaches 0.0. A persistent series is trend reinforcing, which means the direction (up or down compared to the last value) of the next value is more likely the same as current value. The strength of trend increases as H approaches 1.0. Most economic and financial time series are persistent with H>0.5.

In time series forecasting, the first question we want to answer is whether the time series under study is predictable. If the time series is random, all methods are expected to fail. We want to identify and study those time series having at least some degree of predictability. We know that a time series with a large Hurst exponent has strong trend, thus it's natural to believe that such time series are more predictable than those having a Hurst exponent close to 0.5. In this paper we use neural networks to test this hypothesis.

Neural networks are nonparametric universal function approximators [9] that can learn from data without assumptions. Neural network forecasting models have been widely used in financial time series analysis during the last decade [10],[11],[12]. As universal function approximators, neural networks can be used for surrogate predictability. Under the same conditions, a time series with a smaller forecasting error than another is said to be more predictable. We study the Dow-Jones index daily return from Jan. 2, 1930 to May. 14, 2004 and calculate the Hurst exponent of each period of 1024 trading days. We select 30 periods with large Hurst exponents and 30 periods with Hurst exponents close to random series, and then we use these data to train neural networks. We compare forecasting errors for these two groups and find that the errors are significantly different. This research is done using Matlab. All Matlab programs generating result for this paper can be downloaded from www.arches.uga.edu/~qianbo/research.

The remainder of the paper is organised as follows: Section 2 describes the Hurst exponent in detail. Section 3 then describes the monte carlo simulation process we used to generate data with similar structure to the financial series of interest to us. Section 4 describes a scramble test that we conducted to help verify that there is structure in the series due to the order of samples. Section 5 describes neural networks and their use to verify that sequences with larger values of the Hurst exponent can be more accurately learned and predicted than those with lower Hurst exponent values. Finally, the paper is concluded in section 6.

2. Hurst exponent and R/S analysis

The Hurst exponent can be calculated by rescaled range analysis (R/S analysis). For a time series, $X = X_1, X_2, \ldots Xn$, R/S analysis method is as follows:

(1) Calculate mean value m.

$$m = \frac{1}{n}\sum_{i=1}^{n} X_i$$

(2) Calculate mean adjusted series Y

$$Y_t = X_t - m, \quad t = 1, 2, \ldots, n$$

(3) Calculate cumulative deviate series Z

$$Z_t = \sum_{i=1}^{t} Y_i, \quad t = 1, 2, \ldots, n$$

(4) Calculate range series R

$$R_t = max(Z_1, Z_2, \ldots, Z_t) - min(Z_1, Z_2, \ldots, Z_t)$$
$$t = 1, 2, \ldots, n$$

(5) Calculate standard deviation series S

$$S_t = \sqrt{\frac{1}{t}\sum_{i=1}^{t}(X_i - u)^2} \quad t = 1, 2, \ldots, n$$

Here u is the mean value from X_1 to X_t.

(6) Calculate rescaled range series (R/S)

$$(R/S)_t = R_t/S_t \quad t = 1, 2, \ldots, n$$

Note $(R/S)_t$ is averaged over the regions $[X_1, X_t]$, $[X_{t+1}, X_{2t}]$ until $[X_{(m-1)t+1}, X_{mt}]$ where m=floor(n/t). In practice, to use all data for calculation, a value of t is chosen that is divisible by n.

Hurst found that (R/S) scales by power-law as time increases, which indicates

$$(R/S)_t = c*t^H$$

Here c is a constant and H is called the Hurst exponent. To estimate the Hurst exponent, we plot (R/S) versus t in log-log axes. The slope of the regression line approximates the Hurst exponent. For $t<10$, $(R/S)_t$ is not accurate, thus we shall use a region of at least 10 values to calculate rescaled range. Figure 2.1 shows an example of R/S analysis.

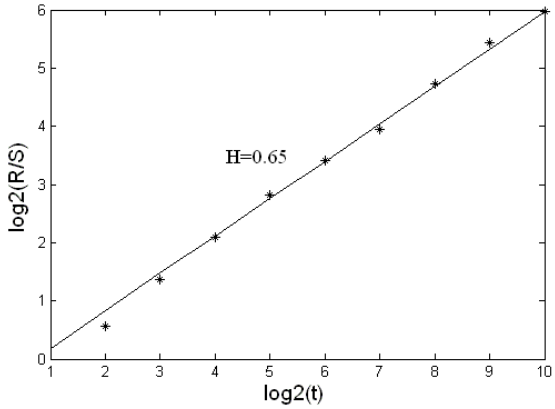

Figure 2.1. R/S analysis for Dow-Jones daily return from 11/18/1969 to 12/6/1973

In our experiments, we calculated the Hurst exponent for each period of 1024 trading days (about 4 years). We use $t = 2^4, 2^5, \ldots, 2^{10}$ to do regression. In the financial domain, it is common to use log difference as daily return. This is especially meaningful in R/S analysis since cumulative deviation corresponds to cumulative return. Figure 2.2 shows the Dow-Jones daily return from Jan. 2, 1930 to May 14, 2004. Figure 2.3 shows the corresponding Hurst exponent for this period. In this period, Hurst exponent ranges from 0.4200 to 0.6804. We also want to know what the Hurst exponent would be for a random series in our condition.

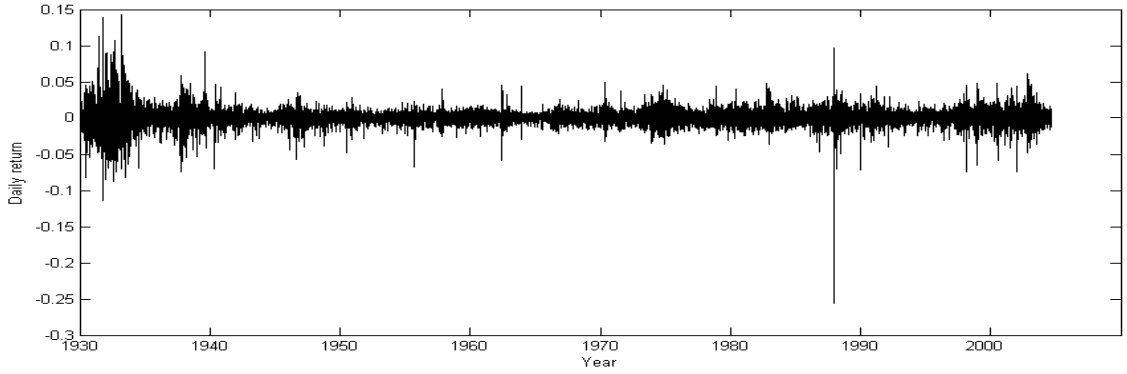

Figure 2.2. Dow-Jones daily return from 1/2/1930 to 5/14/2004

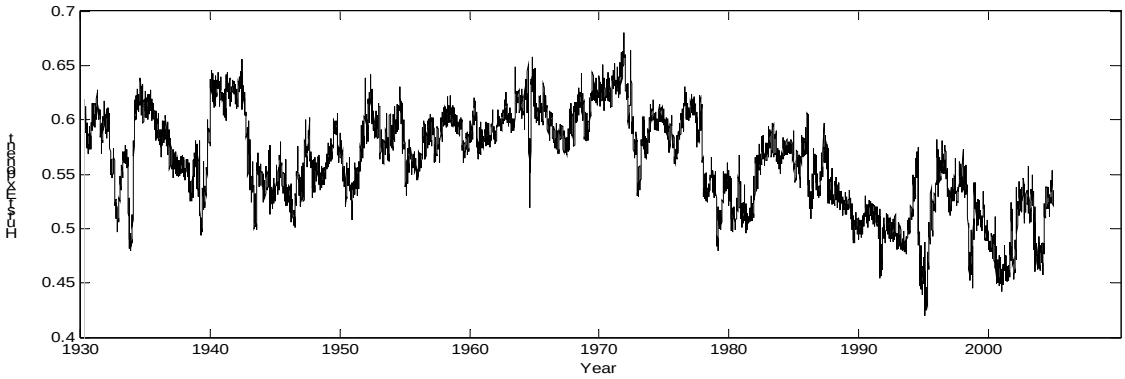

Figure 2.3. Hurst exponent for Dow-Jones daily return from 1/2/1930 to 5/14/2004

3. Monte Carlo simulation

For a random series, Feller [13] gave expected $(R/S)_t$ formula as 3.1.

$$E((R/S)_t) = (n*\pi/2)^{0.50} \quad (3.1)$$

However, this is an asymptotic relationship and is only valid for large t. Anis and Lloyd [14] provided the following formula to overcome the bias calculated from (3.1) for small t:

$$E((R/S)_t = (\Gamma(0.5*(t-1))/(\sqrt{\pi}*\Gamma(0.5*t)))*\sum_{r=1}^{t-1}\sqrt{(t-r)/r} \quad (3.2)$$

For t>300, it is difficult to calculate the gamma function by most computers. Using Sterling's function, formula (3.2) can be approximated by:

$$E((R/S)_t) = (t*\pi/2)^{-0.50}*\sum_{r=1}^{t-1}\sqrt{(t-r)/r} \quad (3.3)$$

Peters [8] gave equation (3.4) as a correction for (3.2).

$$E((R/S)_t) = ((t-0.5)/t)*(t*\pi/2)^{-0.50}*\sum_{r=1}^{t-1}\sqrt{(t-r)/r} \quad (3.4)$$

We calculate the expected (R/S) values for $t=2^4, 2^5,\ldots,2^{10}$ and do least squares regression at significance level α=0.05. Results are shown in table 3.1.

log2(t)	*log2(E(R/S))*		
	Feller	*Anis*	*Peters*
4	0.7001	0.6059	0.5709
5	0.8506	0.7829	0.7656
6	1.0011	0.9526	0.9440
7	1.1517	1.1170	1.1127
8	1.3022	1.2775	1.2753
9	1.4527	1.4345	1.4340
10	1.6032	1.5904	1.5902
Regression Slope (H)	**0.5000** ±5.5511e-016	**0.5436** ± 0.0141	**0.5607** ± 0.0246

Table 3.1. Hurst exponent calculation from Feller, Anis and Peters formula

From table 3.1, we can see that there are some differences between Feller's, Anis' and Peters' formulae. Moreover, their formulae are based on large numbers of data points. In our case, the data is fixed at 1024 points. So what is the Hurst exponent for random series in our case?

Fortunately, we can use Monte Carlo simulation to derive the result. We generate 10,000 Gaussian random series. Each series has 1024 values. We calculate the Hurst exponent for each series and then average them. We expect the average number to approximate the true value. We repeated this process 10 times. Table 3.2 below gives the simulation results.

	Simulated Hurst Exponent	*Standard deviation (Std.)*
1	0.5456	0.0486
2	0.5452	0.0487
3	0.5449	0.0488
4	0.5454	0.0484
5	0.5456	0.0488
6	0.5454	0.0481
7	0.5454	0.0487
8	0.5457	0.0483
9	0.5452	0.0484
10	0.5459	0.0486
Mean	**0.5454**	**0.0485**
Std.	**2.8917e-004**	

Table 3.2. Monte Carlo simulations for Hurst exponent of random series

From table 3.2, we can see that in our situation, the Hurst exponent calculated from Monte Carlo simulations is 0.5454 with standard deviation 0.0485. Our result is very close to Anis' formula. Based on the above simulations, with 95% confidence, the Hurst exponent is in the interval $0.5454 \pm 1.96*0.0485$, which is between 0.4503 and 0.6405. We choose those periods with Hurst exponent greater than 0.65 and expect those periods to be bearing some structure different from random series. However, since these periods are chosen from a large sample (total 17651 periods), we want to know if there exists true structure in these periods, or just by chance. We run a scramble test for this purpose.

4. Scramble test

To test if there exists true structure in the periods with Hurst exponent greater than 0.65, we randomly choose 10 samples from those periods. For each sample, we scramble the series and then calculate the Hurst exponent for this scrambled series. The scrambled series has the same distribution as the original sample except that the sequence is random. If there exists some structure in the sequence, after scrambling the structure will be destroyed and the calculated Hurst exponent should be close to that of a random series. In our experiment, we scramble each sample 500 times and then the average Hurst exponent is calculated. The results are shown in table 4.1 below.

	Hurst exponent after scrambling	*Standard deviation*
1	0.5492	0.046
2	0.5450	0.047
3	0.5472	0.049
4	0.5454	0.048
5	0.5470	0.048
6	0.5426	0.048
7	0.5442	0.051
8	0.5487	0.048
9	0.5462	0.048
10	0.5465	0.052
Mean	**0.5462**	**0.048**

Table 4.1. The average Hurst exponent on 500 scrambling runs

From table 4.1, we can see that the Hurst exponents after the scrambling of samples are all very close to 0.5454 which is the number from our simulated random series. Given this result, we can conclude that there must exist some structure in those periods making them different from random series and that scrambling destroys the structure. We hope this structure can be exploited for prediction. Neural networks, as universal function approximators, provide a powerful tool to learn the underlying structure. They are especially useful when the underlying rules are unknown. We expect neural networks to discover the structure and thus benefit from it. We use neural network prediction error as a measure of predictability. Below we compare prediction errors of the periods with Hurst exponents greater than 0.65 with those between 0.54 and 0.55.

5. Neural networks

In 1943, McClloch and Pitts [15] proposed a computational model simulating neurons. This work is generally thought as the beginning of artificial neural networks research. Rosenblatt [16],[17] popularized the concept of perceptrons and created several perceptron learning rules. However, in 1969, Minsky and Papert [18] pointed that perceptrons cannot solve any non "linearly separable" problems. People knew that multi-layer perceptrons (MLP) can simulate non "linearly separable" functions, but no one knew how to train them. Neural network research was then nearly stopped until 1986. In 1986, Rumelhart [19] used the backpropagation algorithm to train MLP and thus resolved this long obsessed problem among connectionists. Since then neural networks have regained considerable interest from many research fields. Neural networks have become popular in the finance society and the research fund for neural network applications from financial institutions is the second largest [20].

A neural network is an interconnected assembly of simple processing nodes. Each node calculates a summation of weighted inputs and then outputs its transfer function value to other nodes. The Feedforward backpropagation network is the most widely used network paradigm. Using the backpropagation training algorithm, neural network adjusts the weights so that it will minimize the square difference (error) between its observed outputs and their target values. The backpropagation algorithm uses a gradient descent method to find a local minimum on the error surface. It calculates the partial derivative of the square error for each weight. The opposite of these partial derivatives (gradient) gives the direction in which the error decreases most. This direction is called the steepest descent direction. The standard backpropagation algorithm adjusts the weights along the steepest descent direction. Although the error in the steepest descent direction is decreased most rapidly, it usually converges slowly and tends to be stranded due to oscillation. Therefore many backpropagation variants were invented to improve performance by optimizing direction and step size. To name a few, we have backpropagation with momentum, conjugate gradient, Quasi-Newton and Levenberg-Marquardt [21]. After training, we can use the network to do prediction given unseen inputs. In neural network forecasting, the first step is data preparation and pre-processing. After training, we can use the network to do prediction given unseen inputs. In neural network forecasting, the first step is data preparation and pre-processing.

5.1. Data preparation and pre-processing

For Dow-Jones daily return data, we calculated the Hurst exponent for each period of 1024 trading days from 1/2/1930 to 5/14/2004. Among the total of 17651 periods, there are 65 periods with Hurst exponents greater than 0.65 and 1152 periods with Hurst exponents between 0.54 and 0.55. Figure 5.3 below shows the histogram of Hurst exponents for all periods.

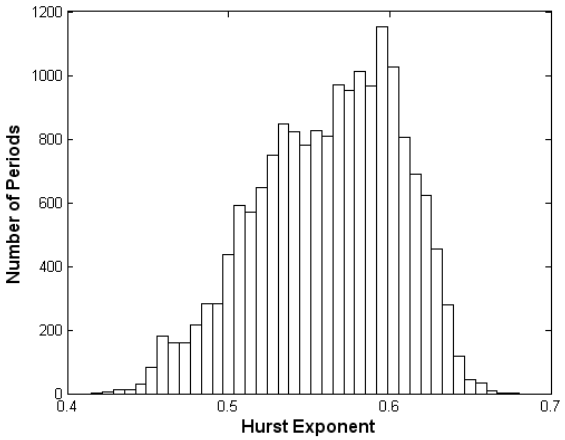

Figure 5.3. Histogram of all calculated Hurst exponents

We randomly chose 30 periods from those with Hurst exponent greater than 0.65 and 30 periods from those with Hurst exponent between 0.54 and 0.55. These two groups of 60 samples constituted our initial data set.

Given a time series x_1, x_2, …, x_i, x_{i+1}, how do we construct a vector $\mathbf{X_i}$ from x_1, x_2, …, x_i to predict x_{i+1}? Taken's theorem [22] tells us that we can reconstruct the underlying dynamical system by time-delay embedding vectors $\mathbf{X_i} = (x_i, x_{i+\tau}, x_{i+2\tau},\ldots,x_{i+(d-1)\tau})$ if we have appropriate d and τ. Here d is called the embedding dimension and τ is called the separation. Using the auto-mutual information and false nearest neighbour methods [23], we can estimate d and τ. We used the TSTOOL [24] package to run the auto-mutual information and false nearest neighbour methods for our 60 data sets. Separations of all data sets are suggested to be 1 by auto-mutual information method. This is consistent with our intuition since we have no reason to use separated values. As for embedding dimension, our data sets are suggested to be from 3 to 5. We shall examine this later.

After building the time-delay vector Xi and target value x_{i+1}, we normalized the inputs Xi and output x_{i+1} to mean 0 and standard deviation 1. We had no need to normalize the output to a limited range, say –0.85 to 0.85, to avoid saturation because we used a linear transfer function in the output layer.

We used a commonly used approach to deal with the over-fitting problem in neural networks. We split the data set to three parts for training, validation and testing. The training data are used to adjust the weights through error backpropagation. The validation data are used to stop training when the mean square error in the validation data increases. The network's prediction performance is judged by the testing data. We used the first 60% of the data for training, the following 20% for validation and the last 20% for testing. In this way, we can have more confidence using the final network model for prediction since the forecasting data follow the testing data directly.

5.2 Neural network construction

Although neural networks are universal function approximators, we still need to pay much attention to their structure. How many layers should we use? How many nodes should we include in each layer? Which learning algorithm should we choose? In practice, most applications use one hidden layer since there is no apparent advantages for multi-hidden-layer networks over single-hidden-layer networks. Thus we used a single hidden layer network structure in our study. For learning algorithm, we tested the Levenberg-Marquardt, conjugate gradient method, and the backpropagation with momentum algorithms. We find that Levenberg-Marquardt beats the other algorithms consistently in our test samples. We chose the Levenberg-Marquardt learning algorithm with the sigmoid transfer function in the hidden layer and a linear transfer function in the output layer. Now we need to determine the embedding dimension and the number of hidden nodes. A heuristic rule to determine the number of hidden nodes is that the total degrees of freedom of a network should equal one and half of the square root of the total data numbers. Based on this rule, we have the following equation:

(#input nodes+1)(#hidden nodes) + (#hidden nodes+1)*(#output nodes) = 1.5sqrt(#data)* (5.2.1)

In equation (5.2.1), 1 is for bias node. For dimension 3, we have:

(3+1)(#hidden nodes)+(#hidden nodes+1) = 1.5sqrt(1024)*

The solution for (#hidden nodes) is 10. Similarly, we find that the number of hidden nodes for dimensions 4 and 5 are 8 and 7 respectively. For each dimension, we examine 5 network structures with the number of hidden nodes adjacent to the number suggested. For example 8, 9, 10, 11, 12 hidden nodes for dimension 3. We randomly choose 5 periods from each group (the group with Hurst exponents greater than 0.65 and the group with Hurst exponents between 0.54 and 0.55) to train each network. Each network is trained 100 times, and then the minimum NRMSE (Normalized Root Mean Square Error) is recorded. NRMSE is defined as:

$$NRMSE = \frac{\sqrt{\sum_i (O_i - T_i)^2}}{\sqrt{\sum_i (T_i - \bar{T})}} \qquad (5.2.2)$$

In (5.2.2), O is output value and T is target value. NRMSE gives a performance comparison to the mean prediction method. If we always use the mean value to do prediction, NRMSE will be 1. NRMSE is 0 when all predictions are correct.

Table 5.1-5.3 gives the training results for various network structures.

	Dimension 3, Hidden nodes						
	8	*9*	*10*	*11*	*12*	*MIN*	*Std.*
1	0.9572	0.9591	0.9632	0.9585	**0.9524**	0.9524	0.0039
2	0.9513	0.9531	0.9560	**0.9500**	0.9523	0.9500	0.0023
3	0.9350	0.9352	0.9332	**0.9328**	0.9359	0.9328	0.0013
4	0.9359	0.9426	0.9383	0.9351	**0.9313**	0.9313	0.0042
5	0.9726	0.9686	0.9652	0.9733	**0.9647**	0.9647	0.0040
6	0.9920	0.9835	0.9892	**0.9793**	0.9931	0.9793	0.0059
7	0.9831	0.9813	**0.9725**	0.9845	0.9825	0.9725	0.0048
8	0.9931	0.9852	**0.9832**	0.9877	0.9907	0.9832	0.0040
9	**0.9684**	0.9790	0.9773	0.9815	0.9862	0.9684	0.0066
10	**0.9926**	1.0044	1.0047	1.0014	1.0039	0.9926	0.0051
Mean	**0.9681**	0.9692	0.9683	0.9684	0.9693	0.9627	0.0042
Std.	0.0225	0.0215	0.0221	0.0232	0.0255	0.0207	0.0015

Table 5.1. NRMSE for dimension 3 with hidden nodes 8, 9, 10, 11 and 12

	Dimension 4, Hidden nodes						
	6	*7*	*8*	*9*	*10*	*MIN*	*Std.*
1	0.9572	0.9572	**0.9557**	0.9558	0.9633	0.9557	0.0031
2	0.9534	0.9554	0.9523	**0.9518**	0.9574	0.9518	0.0023
3	**0.9373**	0.9406	0.9414	0.9437	0.9404	0.9373	0.0023
4	0.9419	0.9471	0.9392	**0.9332**	0.9376	0.9332	0.0052
5	0.9691	0.9678	**0.9597**	0.9669	0.9662	0.9597	0.0037
6	0.9907	0.9939	**0.9836**	0.9948	0.9876	0.9836	0.0046
7	0.9902	0.9816	0.9872	**0.9766**	0.9855	0.9766	0.0053
8	0.9852	0.9842	**0.9802**	0.9865	0.9878	0.9802	0.0029
9	0.9809	0.9729	0.9722	**0.9669**	0.9741	0.9669	0.0050
10	**0.9916**	0.9957	0.9991	1.0025	0.9959	0.9916	0.0041
Mean	0.9698	0.9696	**0.9671**	0.9679	0.9696	0.9637	0.0038
Std.	0.0210	0.0193	0.0204	0.0225	0.0203	0.0196	0.0012

Table 5.2. NRMSE for dimension 4 with hidden nodes 6, 7, 8, 9 and 10

	Dimension 5, Hidden nodes						
	5	*6*	*7*	*8*	*9*	*MIN*	*Std.*
1	0.9578	**0.9560**	0.9617	0.9589	0.9622	0.9560	0.0026
2	0.9441	0.9466	0.9456	0.9456	**0.9427**	0.9427	0.0015
3	0.9410	**0.9395**	0.9449	0.9428	0.9396	0.9395	0.0023
4	0.9546	0.9453	0.9414	**0.9409**	0.9479	0.9409	0.0056
5	0.9659	**0.9501**	0.9671	0.9653	0.9653	0.9501	0.0071
6	0.9906	0.9919	0.9898	0.9901	**0.9891**	0.9891	0.0010
7	**0.9803**	0.9819	0.9805	0.9840	0.9837	0.9803	0.0017
8	**0.9912**	0.9980	1.0009	0.9991	1.0049	0.9912	0.0050
9	0.9770	0.9747	0.9742	0.9761	**0.9689**	0.9689	0.0032
10	**0.9909**	0.9984	0.9975	0.9977	0.9951	0.9909	0.0031
Mean	0.9693	**0.9682**	0.9704	0.9701	0.9699	0.9650	0.0033
Std.	0.0194	0.0233	0.0220	0.0226	0.0227	0.0217	0.0020

Table 5.3. NRMSE for dimension 5 with hidden nodes 5, 6, 7, 8, and 9

From table 5.1 to 5.3, we can see that the differences of NRMSE for nodes within each dimension are very small. The number of hidden nodes with minimum average NRMSE for dimension 3, 4, 5 are 8, 8 and 6 respectively. Thus we use 8, 8, and 6 hidden nodes network for dimension 3, 4, and 5 to do prediction. Each network is trained 100 times and the minimum NRMSE is recorded. Final NRMSE is the minimum of 3 dimensions. Table 5.4 gives NRMSE of our initial 60 samples for two groups.

	H>0.65	*0.55>H> 0.54*		*H>0.65*	*0.55>H> 0.54*
1	0.9534	0.9863	16	0.93	0.9747
2	0.9729	0.9784	17	0.9218	0.9738
3	0.9948	0.9902	18	0.9256	0.9635
4	0.9543	0.9754	19	0.9326	0.957
5	0.9528	0.9773	20	0.937	0.9518
6	0.9518	0.9477	21	0.9376	0.9542
7	0.9466	0.9265	22	0.9402	0.9766
8	0.9339	0.9598	23	0.9445	0.9901
9	0.9299	0.9658	24	0.9498	0.9777
10	0.9435	0.9705	25	0.948	0.9814
11	0.9372	0.9641	26	0.9486	0.9778
12	0.9432	0.9824	27	0.9451	0.9968
13	0.9343	0.9557	28	0.9468	0.9966
14	0.9338	0.9767	29	0.9467	0.9977
15	0.9265	0.9793	30	0.9542	0.9883
Mean	**0.9439**	**0.9731**			
Std.	**0.0145**	**0.0162**			

Table 5.4. NRMSE for two groups

We ran the unpaired student's t test for the null hypothesis that the mean values of the two groups are equal. After calculation, the t statistic is 7.369 and p-value is 7.0290e-010. This indicates the two means are significantly different and the chance of equality is essentially 0. This result confirms that the time series with larger Hurst exponent can be predicted more accurately.

6. Conclusion

In this paper, we analyze the Hurst exponent for all 1024-trading-day periods of the Dow-Jones index from Jan.2, 1930 to May 14, 2004. We find that the periods with large Hurst exponents can be predicted more accurately than those with H values close to random series. This suggests that stock markets are not totally random in all periods. Some periods have strong trend structure and this structure can be learnt by neural networks to benefit forecasting.

Since the Hurst exponent provides a measure for predictability, we can use this value to guide data selection before forecasting. We can identify time series with large Hurst exponents before we try to build a model for prediction. Furthermore, we can focus on the periods with large Hurst exponents. This can save time and effort and lead to better forecasting.

References:

[1] H.E. Hurst, Long-term storage of reservoirs: an experimental study, *Transactions of the American society of civil engineers, 116,* 1951, 770-799.

[2] B.B. Mandelbrot & J. Van Ness, Fractional Brownian motions, fractional noises and applications, *SIAM Review, 10,* 1968, 422-437

[3] B. Mandelbrot, *The fractal geometry of nature* (New York: W. H. Freeman, 1982).

[4] C.T. May, *Nonlinear pricing : theory & applications* (New York : Wiley, 1999)

[5] M. Corazza & A.G. Malliaris, Multi-Fractality in Foreign Currency Markets, *Multinational Finance Journal, 6*(2), 2002, 65-98.

[6] D. Grech & Z. Mazur, Can one make any crash prediction in finance using the local Hurst exponent idea? *Physica A: Statistical Mechanics and its Applications, 336,* 2004, 133-145

[7] E.E. Peters, *Chaos and order in the capital markets: a new view of cycles, prices, and market volatility* (New York: Wiley, 1991).

[8] E.E. Peters, *Fractal market analysis: applying chaos theory to investment and economics* (New York: Wiley, 1994).

[9] K. Hornik, M. Stinchcombe & H. White, Multilayer feedforward networks are universal approximators, *Neural networks, 2*(5), 1989, 259-366

[10] S. Walczak, An empirical analysis of data requirements for financial forecasting with neural networks, *Journal of management information systems, 17*(4), 2001, 203-222

[11] E. Gately, *Neural networks for financial forecasting* (New York: Wiley, 1996)

[12] A. Refenes, *Neural networks in the capital markets* (New York: Wiley, 1995)

[13] W. Feller, The asymptotic distribution of the range of sums of independent random variables, *The annals.of mathematical statistics, 22,* 1951, 427-432

[14] A.A. Anis & E.H. Lloyd, The expected value of the adjusted rescaled hurst range of independent normal summands, *Biometrika, 63,* 1976, 111-116

[15] W. McCulloch and W. Pitts, A logical calculus of the ideas immanent in nervous activity, *Bulletin of Mathematical Biophysics*, 7, 1943,:115 - 133.

[16] F. Rosenblatt, The Perceptron: a probabilistic model for Information storage and organization in the brain, *Psychological Review, 65*(6), 1958, 386-408.

[17] F. Rosenblatt,, *Principles of neurodynamics*, (Washington D.C.: Spartan Press, 1961).

[18] M. Minsky & S. Papert, *Perceptrons* (Cambridge, MA: MIT Press, 1969)

[19] D.E. Rumelhart, G.E. Hinton & R.J. Williams, Learning internal representations by error propagation, in *Parallel distributed processing, 1,* (Cambridge, MA: MIT Press, 1986)

[20] J. Yao, C.L. Tan & H. Poh, *Neural networks for technical analysis: a study on LKCI, International journal of theoretical and applied finance, 2*(3), 1999, 221-241

[21] T. Masters, *Advanced algorithms for neural networks : a C++ sourcebook* (New York: Wiley, 1995)

[22] F. Takens, *Dynamical system and turbulence, lecture notes in mathematics, 898(Warwick 1980),* edited by A. Rand & L.S. Young (Berlin: Springer, 1981)

[23] A.S. Soofi & L. Cao, *Modelling and forecasting financial data: techniques of nonlinear dynamics* (Norwell, Massachusetts: Kluwer academic publishers, 2002)

[24] C. Merkwirth, U. Parlitz, I. Wedekind & W. Lauterborn, TSTOOL user manual, http://www.physik3.gwdg.de/tstool/manual.pdf, 2002