Appl Intell (2007) 26:25–33
DOI 10.1007/s10489-006-0001-7

Stock market prediction with multiple classifiers

Bo Qian · Khaled Rasheed

Published online: 28 November 2006

Abstract Stock market prediction is attractive and challenging. According to the efficient market hypothesis, stock prices should follow a random walk pattern and thus should not be predictable with more than about 50 percent accuracy. In this paper, we investigated the predictability of the Dow Jones Industrial Average index to show that not all periods are equally random. We used the Hurst exponent to select a period with great predictability. Parameters for generating training patterns were determined heuristically by auto-mutual information and false nearest neighbor methods. Some inductive machine-learning classifiers—artificial neural network, decision tree, and *k*-nearest neighbor were then trained with these generated patterns. Through appropriate collaboration of these models, we achieved prediction accuracy up to 65 percent.

Keywords Stock market prediction · Efficient market hypothesis · Hurst exponent · Machine learning · Model ensemble

1 Introduction

Stock market prediction is very difficult. The efficient market hypothesis [1, 2] states that the current market price fully reflects all available information. This implies that past and current information is immediately incorporated into stock prices, thus price changes are merely due to new information or “news”, and independent of existing information. Since news in nature happens randomly and is unknowable in the present, stock prices should follow a random walk pattern and the best bet for the next price is current price. If this hypothesis is true, then any attempts to predict the market will be fruitless.

The random walk model has been tested extensively in various markets. The results are mixed and sometimes contradictory. In general, many early works [3–5] support the random walk model. Jensen [6] claims that “. . .there is no other proposition in economics which has more solid empirical evidence supporting it than the Efficient Market Hypothesis”. However, most recent studies [7–10] on stock markets reject the random walk behavior of stock prices. In a review, Fama [2] even states that the efficient market hypothesis surely must be false.

Although numerous reports give evidence that stock prices are not purely random, all agree that the behavior of stock prices is approximately close to the random walk process. Thus the stock market should not be predictable much past 50% [11]. Degrees of accuracy of 56% hit rate in the predictions are often reported as satisfying results for stock predictions [12, 13].

Besides the efficient market hypothesis and the random walk hypothesis, there are two schools of thought regarding stock market predictions—technical analysis and fundamental analysis. Fundamental analysis examines a company’s financial conditions, operations, and/or macroeconomic indicators to derive the intrinsic value of its common stock. Fundamental analysts will buy/sell if the intrinsic value is greater/less than the market price; however, the proponents of the efficient market hypothesis argue that the intrinsic value of a stock is always equal to its current price. Technical analysis, on the other hand, is a study of the market itself.

B. Qian (✉)
Countrywide Financial Corporation, 4500 Park Granada,
MS CH-21A, Calabasas, CA 91302, USA
e-mail: bo_qian@countrywide.com

K. Rasheed
Department of Computer Science, The University of Georgia,
Athens, GA 30602, USA
e-mail: khaled@cs.uga.edu

Technical analysts believe market action tells everything, so price (open, high, low, and close) and trading volume time series are enough for prediction tasks.

There are three key assumptions in technical analysis. 1. Market action (price, volume) reflects everything. 2. Prices move in trends. 3. History repeats itself. It is interesting to see that both efficient market hypothesis and technical analysis have one similar assumption, yet the assumption leads to totally different conclusions. The key point is whether the market price fully and immediately reflects new information. Technical analysts think stock prices move slowly in response to the news. Since market driving forces (i.e., human psychologies) hardly change, the prices show long-lived trends that are recurrent and predictable.

Technical analysts use tools, such as charts, technical indicators, Dow theory, Gann lines, and Elliot waves, to exploit recurring patterns. "Head-and-Shoulders" and "Double Tops and Bottoms (M and W formation)" are two examples for charting patterns to indicate market reversal. Trading rules such as "Buy when short-term moving average upwardly crosses long-term moving average and sell when they downwardly cross" and "Buy when RSI (Relative Strength Index) is below 20 and sell when it is over 80" have been tested and are often used on the trading floor.

One big problem with these technical techniques is "self-destructing". Once a profitable trading strategy becomes well known, the opportunity will disappear quickly if all traders take the same buy or sell action. Thus a successful strategy should not be easily copied. Due to the "regime shifting" [14] character of the market, a successful trading strategy must also be dynamic and self-adaptive. Artificial neural networks are nonparametric universal function approximators [15] that can learn hidden patterns from data without assumptions. They are ideal for stock market prediction. Neural network forecasting models have been widely used in financial time series analyses during the last decade [16–18]. Given a data set, even though there might not be an exploitable forecasting relation, we can still get a "good" model by conducting extensive searches. This is called "data snooping" [19]. Since degrees of accuracy of 56% hit rate in the predictions are often reported as satisfying results, data snooping is a very serious problem in financial market prediction. It will become worse if we abuse the powerful regression ability of neural networks. In our research, we use two additional inductive machine-learning methods—decision tree and k-nearest neighbor—to reduce the risk of data snooping. Through the appropriate ensemble of neural network, decision tree, and k-nearest neighbor classifiers, we can improve both the accuracy and stability of our model. This research was conducted using Matlab. All Matlab programs results for this paper can be downloaded from www.arches.uga.edu/∼qianbo/research.

2 Data selection

In this study, we examine the Dow Jones Industrial Average (DJIA) index. This index has a long history of data and it has been widely referred to and studied. Numerous papers have discussed forecasting for the DJIA index. However, to our best knowledge, no one has proposed a method as to how to choose a date period for better prediction. The date period is usually chosen arbitrarily without any explanations. In this paper, we use the Hurst exponent to select a period with great predictability to perform further investigation.

The Hurst exponent, proposed by Hurst [20] for use in fractal analysis [21, 22], has been applied to many research fields. It has recently become popular in the finance community [23–25] largely due to Peters' work [26, 27]. The Hurst exponent provides a measure for the long-term memory and fractality of a time series. Since it is robust with few assumptions about the underlying systems, it has broad applicability for time series analyses. The values of the Hurst exponent, H, range between 0 and 1, and based on its value, a time series can be classified into one of 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, by contrast, is trend reinforcing, which means the direction (up or down) of the next value is likely the same as that of the current value. The strength of this trend increases as H approaches 1.0. In [28], we show that the time series with a large H are more predictable than those with an H that is close to 0.5.

The Hurst exponent can be estimated by rescaled range analysis (R/S analysis). We calculated the Hurst exponent for each period of 1024 trading days (about 4 years) from 1 January 1930 to 14 May 2004. We chose a length of 1024 days for two reasons. One reason is that according to Peters [27], the Dow Jones index has a four-year cycle. The other reason is that it provides enough data for a test, yet it does not suffer much from the Time Series Recency Effect [29].

The Time Series Recency Effect is a phenomenon in building time series models that using data closer in time to the forecasted data usually produce better models. Figure 1 shows the Dow Jones daily return from 2 January 1930 to 14 May 2004. As is common in the finance community, we used the log difference to calculate the daily return. Figure 2 shows the corresponding Hurst exponent for this period. In this period, the Hurst exponent ranges from 0.4200 to 0.6804. It is interesting to see that in recent years, the Hurst exponent has moved closer to 0.5, which is the Hurst exponent value of a random series. This implies that the market has become

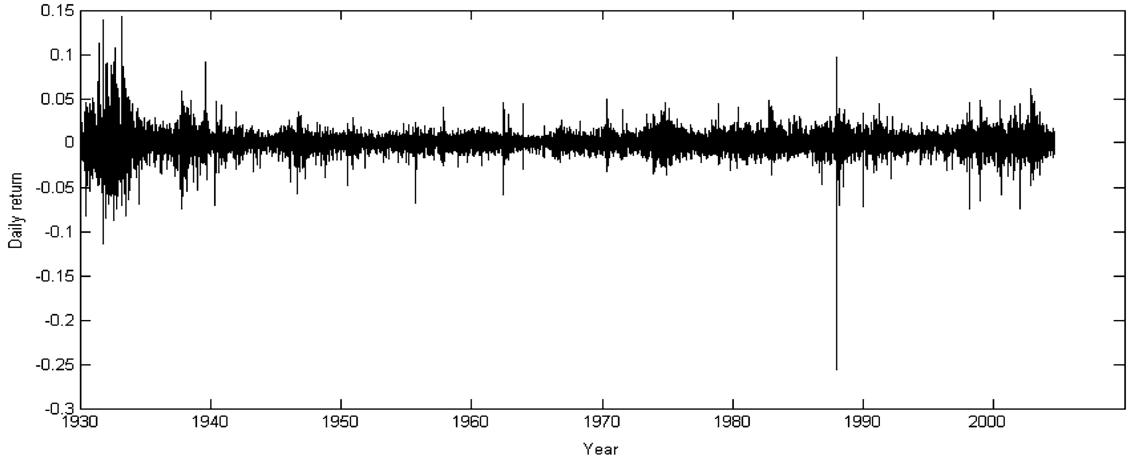

Fig. 1 Dow Jones daily returns from 1/2/1930 to 5/14/2004

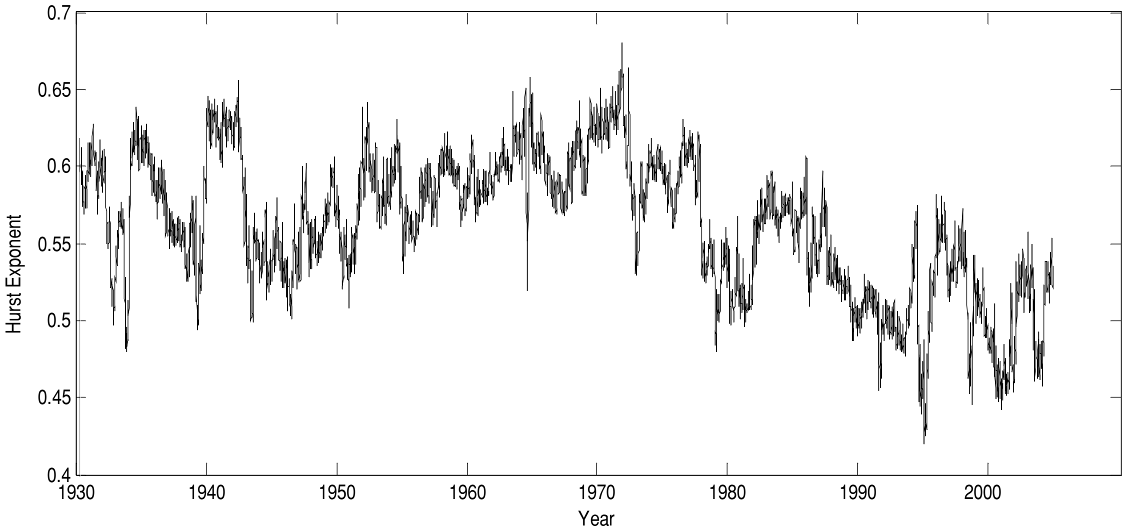

Fig. 2 Hurst exponent for Dow Jones daily return from 1/2/1930 to 5/14/2004

more efficient in recent years. It is not a good idea to use any of the more random periods to make a prediction. The largest Hurst exponent value (0.6804) was reached on 4 June 1973, so for our experiment, we chose the 4-year period from 4 June 1969 to 4 June 1973. We chose this particular 4-year period since we were more interested in looking at a 4-year period rather than examining 1024 trading days. There are 1010 trading days in this 4-year period.

3 Training pattern generation

Now that we had a Dow Jones daily return time series from 4 June 1969 to 4 June 1973, we wanted to predict a return sign (up or down) based on previous returns. How do we generate training patterns to be used in machine learing models? The sliding window technique is a common method to generate training patterns from time series. In this method, d inputs and a single output window slide over the whole time series. For example, given a time series $x_1, x_2, \ldots x_{N-1}, x_N$, under the sliding window method, input vectors $\boldsymbol{X}_1 = (x_1, x_2, x_3, \ldots x_d)$, $\boldsymbol{X}_2 = (x_2, x_3, x_4, \ldots x_{d+1}) \ldots$ are generated. However, input vectors can also be $\boldsymbol{X}_1 = (x_1, x_3, x_5, \ldots x_{2d-1})$, $\boldsymbol{X}_2 = (x_2, x_4, x_6, \ldots x_{2d}) \ldots$. We need to choose a vector size d and separation τ (1 day, 2 days, etc.).

Since more and more evidence [26, 27, 30] shows that stock markets are chaotic non-linear dynamic systems, we use heuristics from chaos theory to determine d and τ. Taken's theorem [31] tells us that we can reconstruct the underlying dynamical system by time-delay embedding vectors $\boldsymbol{X}_i = (x_i, x_{i+\tau}, x_{i+2\tau}, \ldots, x_{i+(d-1)\tau})$ if we have the appropriate d and τ. Here d is called the embedding dimension and τ is called the time-delay or separation. Using the auto-mutual information and Cao's false nearest neighbor methods [32, 33], we can estimate d and τ. Frank etc. [34] has shown that

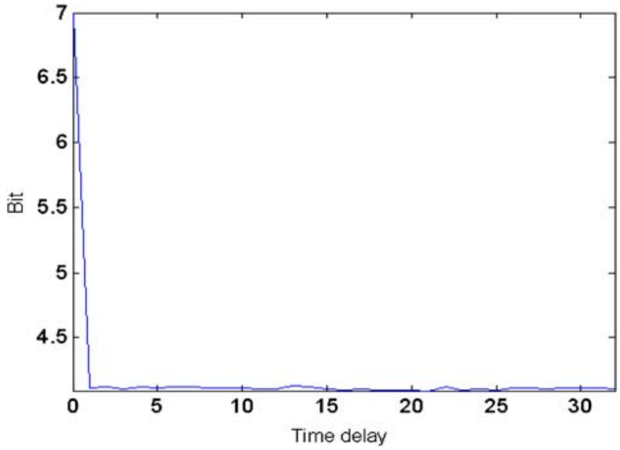

Fig. 3 Auto-mutual information for Dow-Jones daily returns from 6/4/1969 to 6/4/1973

the estimated d and τ by these methods are good heuristics to generate training patterns for prediction. We used the TSTOOL [35] package to run the auto-mutual information and false nearest neighbour methods for our data. Figure 3 gives the auto-mutual information for the Dow Jones daily return from 4 June 1969 to 4 June 1973. From this figure, we can see that the first minimum of the auto-mutual information is at 1, so we chose separation $\tau = 1$. Figure 4 gives the result of Cao's false nearest neighbor method for minimum embedding dimension estimation. E1(d) measures the variation of the average relative change of distances between pairs of neighbors when the dimension increases from d to $d + 1$. The suggested minimum embedding dimension is value d when the increment of E1(d) slows down and begins to saturation. There is a kink in the graph at dimension 4, so we chose embedding dimension $d = 4$.

Since $\tau = 1$ and $d = 4$ are suggested, we used four continuous daily returns to predict the direction of the fifth day. Then 1010 training patterns were generated. Each training pattern had the form $(r_{i-3}r_{i-2}r_{i-1}r_i, d_{i+1})$, where r_i was the daily return on day i and d_{i+1} is 1 if $r_{i+1} > 0$ and –1 if $r_{i+1} < 0$. Since the four inputs $r_{i-3,}r_{i-2,}r_{i-1}$ and r_i are in the same scale, it is not necessary to do any normalization. These 1010 patterns formed the data set for our machine learning classifiers. We used the last 20% (202 patterns) of the data as out-of-sample data to judge the performance of the final models. These out-of-sample data were never used in any model generation or evaluation processes. The remaining 80% (808 patterns) of the data, as in-sample data, was used to develop models.

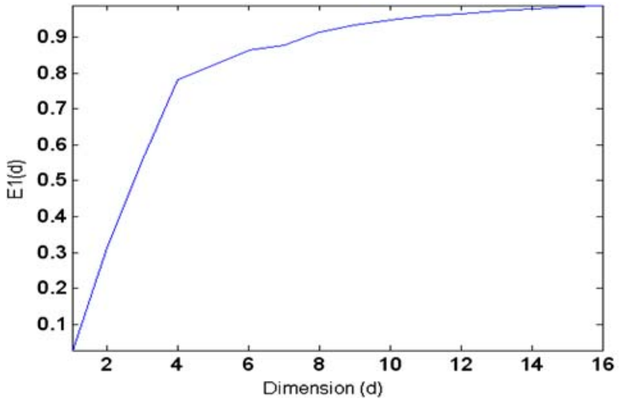

Fig. 4 Minimum embedding dimension for Dow -Jones daily returns from 6/4/1969 to 6/4/1973 using Cao's method

4 Classifiers

In our experiments, artificial neural network (ANN), k-nearest neighbor (kNN), and decision tree (DT) learning algorithms were used for prediction tasks. Experimental settings for each algorithm are described below.

Artificial neural network (ANN)

We used a feed-forward one-hidden-layer network. 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. The Levenberg-Marquardt learning algorithm [36] was designed to approach second-order training speed without having to compute the second-derivative matrix (Hessian matrix). It uses a first-order derivative matrix (Jacobian matrix) to approximate the Hessian matrix. The Jacobian matrix can be computed through a standard backpropagation technique [37] that is much less complex than computing the Hessian matrix. The Levenberg-Marquardt algorithm usually converges most quickly and with the lowest mean square errors when the network is moderately sized (up to a few hundred weights). In the training phase, target values were 1 and –1, representing the up and down direction respectively. In the prediction phase, outputs greater than 0 were considered up and those less than 0 were considered down.

k-nearest neighbor (kNN)

The k-nearest neighbor [38] algorithm is an instance-based learning algorithm. It uses similar cases to predict unknown cases. In our experiment, we used Euclidean distance to measure similarity. Given an unknown case, the k most similar cases (neighbors) in the training data are found. The unknown case is then assigned to a category (up or down) matching that of the majority of these k cases. In the situation of even cases, we break even according to the average distance.

Table 1 Error rates (%) of the neural network cross-validation for 1 to 10 hidden nodes

	1	2	**3**	4	5	6	7	8	9	10
Minimum	39.25	39.39	39.36	38.64	39.48	39.26	39.88	39.00	39.62	39.49
Maximum	41.58	42.58	40.85	41.88	40.86	41.98	41.23	41.10	42.22	41.36
Average	40.51	40.62	**39.91**	40.02	40.23	40.74	40.73	40.22	40.78	40.29
Std. dev.	0.7908	1.0793	0.5352	0.9838	0.4607	0.9251	0.4647	0.7037	0.6565	0.6246

Table 2 Error rates (%) of the *k*-nearest neighbor cross-validation for *k* from 1 to 10

	1	2	3	4	**5**	6	7	8	9	10
Minimum	44.19	42.35	41.96	42.74	42.09	42.84	43.21	42.95	42.59	42.45
Maximum	46.55	45.44	45.32	45.80	44.21	46.06	44.94	45.80	44.94	45.19
Average	44.98	44.26	44.20	43.99	**43.36**	44.73	44.32	44.34	43.48	43.87
Std. dev.	0.7719	1.0681	1.0457	1.0551	0.8790	1.0959	0.5392	0.8899	0.6403	0.7645

Table 3 Error rates (%) of the decision tree cross-validation for 1 to 10 maximum impure cases in a leaf

	1	2	3	4	5	6	7	8	9	10
Minimum	42.96	43.47	42.61	43.59	41.97	43.00	43.72	43.32	43.59	43.20
Maximum	44.56	46.06	45.56	48.42	45.67	45.06	45.56	46.06	45.81	46.20
Average	**43.62**	44.40	43.93	44.95	43.63	44.00	44.37	44.47	44.50	44.59
Std. dev.	0.4784	0.8111	0.9442	1.4576	1.0244	0.6343	0.6624	0.9954	0.6714	1.0952

Decision tree (DT)

Decision trees [39] classify an instance by filtering it down a tree from the root node to a leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values (for a discrete attribute) or ranges of values (for a continuous attribute). In our experiment, the CART [40] algorithm was used to build decision trees. The Gini diversity index was used to split tree nodes, and cross-validation was used to prune the tree.

4.1 Parameter determination

Aside from the settings described above, for all learning algorithms, there was still one major parameter to be decided. For the artificial neural network, we tested numbers of hidden nodes from 1 to 10. For the decision tree, we tested the maximum number of allowed impure cases in a leaf from 1 to 10. For the *k*-nearest neighbor, we tested values of *k* from 1 to 10. Since the prediction results of these classifiers are sensitive to data split, we used cross-validation [41] to judge the ability of generalization. Because we used 20% of the data as a test data set, to simulate this partition, we used a 5-fold cross-validation method to evaluate models. For the *k*-nearest-neighbor, the remaining 80% of the data was used as training data. For the neural network and the decision tree, 20% of the data was used as a validation set, and the other 60% as a training set. The validation data set for a neural network was used for early stopping in order to avoid over-fitting, while for a decision tree, this set was used to choose optimal pruned trees. In order to produce different models for each run, we used randomized initial weights in the neural network, while for the *k*-nearest neighbor and decision tree, different training data were generated by a bootstrap technique as suggested in Bagging [42]. For each classifier, we ran a 5-fold cross-validation 10 times. Minimum, maximum, and average classification error rates for each parameter setting were recorded. Tables 1–3 show the 5-fold cross-validation results for the neural network, the *k*-nearest-neighbor, and the decision tree. We chose a parameter setting with a minimum average cross-validation error rate for our following experiments.

4.2 Classifiers ensemble

Research in methodologies and systems for the combination of multiple predictive models has recently become very active. These methods are referred to as ensemble methods in the machine learning community [43]. An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way (typically by weighted or unweighted voting) to classify new examples. The goal of ensemble methods is to generate an overall system that performs better than any individual classifier. In order for an ensemble of classifiers to be more accurate than any of its classifiers, the classifiers must be accurate and diverse [44]. Here accurate means better than random guessing and diverse means the classifiers make uncorrelated errors.

A great deal of research has shown that ensembles are in practice often more accurate than the best of the classifiers they are composed of [45]. Since the accuracy of financial market prediction is usually little above 50%, ensemble methods can be especially useful for financial market prediction. Even a 1% improvement can bring large pecuniary rewards. In addition to performance improvements, combining several different classifiers results in a more stable final system, since it can continue to function even if one underlying classifier fails. This consistency is a very important character of successful systems for investment decisions. We anticipate that ensemble methods will become popular in financial market prediction fields in the near future.

Models that have been derived from different executions of the same learning algorithm are usually called homogeneous. Bagging [42] and boosting [46] are the two of the most famous techniques for constructing ensembles of homogeneous classifiers. Models that have been derived from running different learning algorithms on the same data set are called heterogeneous. Voting and stacking [46] are two effective methods to combine the results of heterogeneous classifiers. In our experiment, we investigated the simplest unweighted voting and the state-of-the-art stacking methods. In unweighted voting, each model outputs a class value and has one vote for the class, and then the output of the ensemble is the class with the most votes. Stacking, also called stacked generalization, is a kind of meta-learning method. In stacking, one meta-learner learns from output generated by a set of base-learners (e.g., classifiers). The meta-learner tries to combine the predictions of the base-learners by learning their biases and correlations. The predictions of the base-learners and the corresponding target values form the training data set for the meta-learner. To classify an input pattern, the base-learners make their predictions on the input pattern first. These predictions are then input to the meta-learner and the pattern is classified by the prediction of the meta-learner. The procedure of stacking proposed by Wolpert [47] is elaborated as follows:

Given a data set $L = \{(\boldsymbol{x}_i, y_i), 1 \leq i \leq I\}$ and J classifiers $C_j (1 \leq j \leq J)$, where $\boldsymbol{x}_i$ is the i-th input vector and y_i is the corresponding target class value, we split L into K nearly equal disjoint folds. Let $L^k (1 \leq k \leq K)$ be the k-th fold and $L^{(-k)}$ be all the other data in L.

1. Generate training data set for the meta-learner.
   For each classifier $C_j (1 \leq j \leq J)$, use $L^{(-k)} (1 \leq k \leq K)$ as training data to build model $M_j^{(-k)}$. Then use the remaining data $L^k(\boldsymbol{x})$ as input for model $M_j{}^{(-k)}$ and get output $L_j{}^k(\tilde{y})$. After training by all K folds, we get output $L_j(\tilde{y}) = \{L_j{}^k(\tilde{y})\ \}\ 1 \leq k \leq K\}$, the set of all predictions from classifier C_j with one prediction for each element in L. Combining the J outputs $L_j(\tilde{y})$ and the corresponding target data $L(y)$, we get the training data set *LM* for the meta-learner:

$$LM = \{(\tilde{y}_{1i}, \tilde{y}_{2i}, \ldots, \tilde{y}_{Ji}, y_i), 1 \leq i \leq I\}$$

2. Build model for meta-learner.
   In this step, use *LM* as training data for the meta-learner to build model M. *LM* consists of the predictions of the base-learners along with the target values, and thus the meta-learner will learn the prediction relationship between the base-learners. This completes the meta-learning process.
3. Build models for classifiers.
   This step is needed for classifying a new instance. In this step, all data in L are used as training data to build a single model $M_j (1 \leq j \leq J)$ for each classifier.
4. Prediction.
   To classify a new instance $\boldsymbol{x}$, we input vector $\boldsymbol{x}$ to models $M_j (1 \leq j \leq J)$. This produces a vector $(\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_J)$. This vector is then input to the meta-learner model M. The output of M is the final classification result for $\boldsymbol{x}$. This completes the procedures proposed by Wolpert.

Ting and Witten [48] suggest using probabilities of the output class instead of a class value $\tilde{y}_{ji}$ in step 1. That is in step 1, if there are T classes, for an input vector $\boldsymbol{x}_i$, model $M_j^{(-k)}$ outputs a vector $\boldsymbol{p}_{ji} = (p_1, p_2, \ldots, p_T)$, the predicted probabilities of $\boldsymbol{x}_i$ belonging to each class. In this case, *LM* is

$$\{LM = (\boldsymbol{p}_{1i,}\boldsymbol{p}_{\boldsymbol{2i},\ldots,}\boldsymbol{p}_{Ji,}y_i), 1 \leq i \leq I\}$$

Since $\Sigma p_i = 1 (1 \leq i \leq T)$, we can use $(p_1, p_2, \ldots, p_{T-1})$ for $\boldsymbol{p}_{ji}$ without any information loss.

We noticed that in step 4, the vector $(\tilde{y}_1, \tilde{y}_2, \ldots, \tilde{y}_J)$ input to the meta-learner model M is generated by models M_j while in step 1, the training data set $(\tilde{y}_{1i}, \tilde{y}_{2i}, \ldots, \tilde{y}_{Ji})$ for model M is generated by models $M_j^{(-k)}$. This assumes that model M_j and models $M_j^{(-k)}$ are equivalent, which may not be a valid assumption for unstable learning algorithms—algorithms whose output models undergo major changes in response to small changes in the training data. Neural networks and decision trees belong to the category of unstable algorithms. In this paper, in addition to standard stacking, we also investigate a variant model $M_j{}'$ as a substitution of M_j in steps 3 and 4. Let f(M, $\boldsymbol{x}$) be the output of model M given input $\boldsymbol{x}$. $M_j{}'$ is defined as

$$f(M_j', x) = \frac{1}{K} \sum_{k=1}^{K} f\big(M_j^{(-k)}, x\big).$$

Table 4 Error rates for neural network, k-nearest-neighbor, and decision tree

	Neural network		k-nearest-neighbor		Decision tree		Mean of
	Mean	Std. dev.	Mean	Std. dev.	Mean	Std. dev.	three classifiers
Test (M_j)	40.29	4.37	42.90	2.87	42.16	2.91	41.79
Prediction (M_j)	41.39	2.51	43.76	2.38	39.21	2.47	41.45
5-fold test ($M_j^{(-k)}$)	38.57	1.32	43.53	1.24	42.63	1.23	41.57
5-fold prediction ($M_j^{(-k)}$)	41.87	1.16	43.81	0.69	38.68	1.43	41.46
5-fold average prediction ($M_j{}'$)	41.04	1.29	43.07	1.28	38.02	1.81	40.71

Table 5 Average error rates for ten randomly chosen periods

	Neural network		k-nearest-neighbor		Decision tree		Mean of
	Mean	Std. dev.	Mean	Std. dev.	Mean	Std. dev.	three classifiers
Test (M_j)	46.87	3.66	47.13	2.39	47.79	2.81	47.26
Prediction (M_j)	47.12	3.17	47.21	2.07	48.28	3.05	47.54
5-fold test ($M_j^{(-k)}$)	47.04	2.94	47.58	2.78	48.09	2.68	47.57
5-fold prediction ($M_j^{(-k)}$)	47.43	3.48	47.39	1.89	48.45	2.77	47.76
5-fold average prediction ($M_j{}'$)	46.99	3.11	47.33	2.30	48.17	2.43	47.50

That is, the output of $M_j{}'$ is the average of models $M_j^{(-k)}(1 \leq k \leq K)$. We think model $M_j{}'$ is a better representation than M_j for training data set generated by models $M_j^{(-k)}$. Notice that the meta-learning procedures (steps 1 and 2) in our stacking method are the same as Wolpert and Ting's method. Our stacking method differs only in classifying new instances. We use model $M_j{}'$ instead of M_j in step 4 and step 3 is unnecessary in our method since $M_j{}'$ is already produced in step 1. Predictions by model $M_j{}'$ and M_j are both input to the meta-learner model M to get a final result for each new instance. We refer to Wolpert and Ting's stacking model as SM and our stacking model as SM′.

5 Running results

5.1 Results for individual classifiers

Using the configuration and parameters discussed above, we ran each classifier 10 times. As previously mentioned, we used the last 20% of the data as a prediction data set to judge the final performance. Of the remaining data, the last 20% was used as test data. For each run, we executed each learning algorithm 5 times to produce 5 models. The model with the smallest error rate for the test data was selected as model M_j. The prediction data was then input to this model. The output of model M_j is the prediction result.

Error rates for both test and prediction data were then recorded. The mean and standard deviation of 10 runs were calculated. For 5-fold cross-validation, we produced one model $M_j^{(-k)}$ for each fold. The prediction data was then input to these 5 models. The average error rate of these 5 models was calculated as a performance measure for the 5-fold prediction. The average prediction of the 5 models was generated as the output of model $M_j{}'$ and the error rate of this 5-fold average prediction was recorded. Meanwhile, the combination of the output of the 5-fold test data for model $M_j^{(-k)}$ was generated as meta-learning training data for stacking purposes. Table 4 shows the test and prediction error rates for each classifier. We can see the prediction error rates for individual classifier are between 38.02% and 43.81%, which translates to a range of hit rates between 56.19% and 61.98%. This result is better than previously reported hit rate of 56% [12, 13]. We believe date period selection is important to stock market prediction and the Hurst exponent provides a good measure for data selection. To confirm this, we randomly chose ten 4-year periods and ran the same procedures as described before. The average Hurst exponent is 0.5325 for these 10 periods. Table 5 gives these results. It is clear that using data with large Hurst exponent generates better prediction accuracy.

From Table 4, we can see that although the neural network had the smallest error rates for the test data, the smallest error rates for the prediction data were achieved by the decision tree. This means the simple model evaluation and selection method did not work well in this case. It serves as an example for the necessity of a combination of multiple classifiers.

5.2 Results for ensembles of classifiers

Table 6 shows the results for stacking and voting. Our 5-fold average prediction method was better than the standard stacking method, although the difference was not significant. Also,

Table 6 Error rates for different ensemble methods

	Stacking		Voting		Consistent voting	
	Mean	Std. dev.	Mean	Std. dev.	Mean	Std. dev.
Prediction output (M_j)	41.49 (SM)	1.56	40.79	1.90	36.12	1.62
5-fold average prediction output ($M_j{}'$)	**40.99** (SM′)	1.17	**39.50**	1.03	**34.64**	1.82

Table 7 Correlation coefficient of prediction

	Prediction (M_j)			5-fold average prediction ($M_j{}'$)		
	ANN	kNN	DT	ANN	kNN	DT
ANN	**0.9281**	0.7853	0.5007	**0.9610**	0.7108	0.4447
kNN	0.7853	**0.8352**	0.5251	0.7108	**0.9854**	0.4903
DT	0.5007	0.5251	**0.8020**	0.4447	0.4903	**0.9854**

Table 8 Number of consistent instances and error rates of consistent voting

	ANN-kNN # (error rate)	ANN-DT # (error rate)	KNN-DT # (error rate)	ANN-kNN-DT # (error rate)
Prediction output	1736 (39.69%)	1532 (37.21%)	1576 (37.25%)	1412 (36.12%)
5-fold average prediction output	1589 (38.39%)	1391 (36.88%)	1334 (35.83%)	**1147 (34.64%**)

the performance of simple unweighted voting was better than stacking. The performance of these ensembles did not exceed the best individual classifier. This may indicate that the errors of classifiers are not independent, but may be highly correlated. We calculated the correlation coefficients of the prediction between each classifier pair. The correlation coefficient is a measure of the strength of the linear relationship (correlation) between two data sets. Its range is between negative and positive one. A correlation coefficient of one indicates a perfect one-for-one movement trend in the data sets, while a coefficient of negative one indicates exactly the opposite movement trend.

We ran each classifier 5 times, and calculated the correlation coefficient between each run. The average of these 10 (5*4/2) correlation coefficients was recorded as a correlation coefficient of the predictions within the classifier. For the correlation coefficients between each classifier, they were recorded as the average of 10 runs. Table 7 shows the correlation coefficient results. From Table 7, we can see that the correlation coefficients between different classifiers were much lower than those within classifiers. However, the correlation coefficients between different classifiers were still high, especially for the neural network and k-nearest neighbor.

We further investigated how many predictions agreed among three classifiers for a total of 2020 (202 instances in prediction set * 10 runs) predictions assuming a 60% accurate prediction. In theory, two classifiers will agree on instances of 52% (0.6*0.6 + 0.4*0.4) and three classifiers will agree on 28% (0.6*0.6*0.6 + 0.4*0.4*0.4) if their predictions are independent. This gives a theoretical number of 1050 agreements for two classifiers and 566 for three classifiers. Table 8 gives the number of agreed-upon predictions for different classifiers in our experiments. We can see the numbers were much higher than those from the theory. This also indicates a high correlation of predictions between classifiers. This may explain why ensembles of stacking and simple voting did not exceed the performance of the best individual classifier. In this paper, we further studied the performance of a consistent voting method—a method that only counts predictions agreed upon by all classifiers. This is appropriate for our stock market prediction application since we can simply take the position of "no trading strategy" if the prediction is not unanimously agreed upon by all classifiers. Table 8 also shows the error rates for the consistent voting of two and three classifiers. From this table, we can see that the performances of all consistent voting results were better than the best individual classifier in the ensemble. We achieved a 34.64% error rate when all three classifiers agreed in the 5-fold average prediction. This means we only missed once in three predictions.

6 Conclusion

In this paper, we attempted to make a prediction for the Dow Jones Industrial Average time series. Using the Hurst exponent and heuristics from chaos theory, we easily achieved a degree of 60% accuracy in our prediction. Since the Hurst exponent provides a measure for predictability, we can use this value to guide data selection before forecasting. Given the thousands of actively traded stocks, we can identify those time series with large Hurst exponents before we try to build models for prediction. This can save time and will lead to superior results. We also investigated ensemble methods to improve prediction performance. Simple voting and stacking ensemble methods did not work well due to high correlations of predictions between classifiers. However, even in this situation, a consistent voting ensemble method improved prediction accuracy from 60% to 65%. Because of the highly competitive nature of financial market predictions, we expect ensembles of multiple classifiers will be popular in this field.

References

1. Fama EF, Fisher L, Jensen M, Roll R (1969) The adjustment of stock price to new information. Int Eco Rev 10(1):1–21
2. Fama EF (1991) Efficient capital markets: II J Fin 46(5):1575–1617
3. Cootner PH (1964) The random character of stock market prices. MIT Press, MA
4. Fama EF (1965) The behaviour of stock market prices. J Bus 38:34–105
5. Alexander SS (1961) Price movements in speculative markets: Trends or random walks. Ind Manage Rev pp 7–26
6. Jensen MC (1978) Some anomalous evidence regarding market efficiency. J Fin Eco 6:95–102
7. Gallagher L, Taylor M (2002) Permanent and temporary components of stock prices: Evidence from assessing macroeconomic stocks. Southern Eco J 69:245–262
8. Lo AW, MacKinlay AC (1997) Stock market prices do not follow random walks. Market Efficiency: Stock Market Behaviour in Theory and Practice 1:363–389
9. Kavussanos MG, Dockery E (2001) A multivariate test for stock market efficiency: The case of ASE Applied Financial Economics 11(5):573–579(7)
10. Kirt CB, Malaikah SJ (1992) Efficiency and inefficiency in thinly traded stock markets: Kuwait and Saudi Arabia. J Bank & Fin 16(1):197–210
11. Walczak S (2001) An empirical analysis of data requirements for financial forecasting with neural networks. J Manag Infor Syst 17(4):203–222
12. Baestaens DJE, van den Bergh WM, Vaudrey H (1996) Market inefficiencies, technical trading and neural networks. In: Dunis C (ed) forecasting financial markets, financial economics and quantitative analysis. John Wiley & Sons, Chichester, England, pp 254–260
13. Tsibouris G, Zeidenberg M (1995) Testing the efficient markets hypothesis with gradient descent algorithms. In: Refenes AP (ed) Neural networks in the capital markets. John Wiley & Sons, Chichester, England, Chap 8, pp 127–136
14. Hellstrom T, Holmstrom K (1998) Predicting the stock market, technical report series IMa-TOM-1997-07, Center of Mathematical Modeling, Malardalen University
15. Hornik K, Stinchcombe M, White H (1989) Multilayer feedforward networks are universal approximators. Neural Net 2(5):259–366
16. Refenes A (1995) Neural networks in the capital markets. Wiley, New York
17. Gately E (1996) Neural networks for financial forecasting. Wiley, New York
18. Zirilli JS (1997) Financial prediction using neural networks. International Thomson Computer Press, UK
19. White H (2000) A reality check for data snooping. Econometrica 68(5):1097–1126
20. Hurst HE (1951) Long-term storage of reservoirs: an experimental study. Trans Amer Soc Civil Engi 116:770–799
21. Mandelbrot BB, Ness JV (1968) Fractional brownian motions, fractional noises and applications. SIAM Rev 10:422–437
22. Mandelbrot B (1982) The fractal geometry of nature. WH Freeman, New York
23. May CT (1999) Nonlinear pricing: theory & applications. Wiley, New York
24. Corazza M, Malliaris AG (2002) Multi-fractality in foreign currency markets. Multinat Fin J 6(2):65–98
25. Grech D, Mazur Z (2004) Can one make any crash prediction in finance using the local Hurst exponent idea?. Physica A: Statistical Mech Appl 336:133–145
26. Peters EE (1991) Chaos and order in the capital markets: a new view of cycles, prices, and market volatility. Wiley, New York
27. Peters EE (1994) Fractal market analysis: applying chaos theory to investment and economics. Wiley, New York
28. Qian B, Rasheed K (2004) Hurst exponent and financial market predictability. In: Proceedings of The 2nd IASTED international conference on financial engineering and applications. Cambridge, MA, USA, pp 203–209
29. Walczak S (2001) An empirical analysis of data requirements for financial forecasting with neural networks. J Manag Infor Syst 17(4):203–222
30. Hsieh DA (1991) Chaos and nonlinear dynamics: application to financial markets. J Fin 46:1839–1877
31. Takens F (1981) Dynamical system and turbulence. In: Rand A, Young Ls (eds) Lecture notes in mathematics, 898(Warwick 1980). Springer, Berlin
32. Cao L (1997) Practical method for determining the minimum embedding dimension of a scalar time series. Physica D 110:43–50
33. Soofi AS, Cao L (2002) Modelling and forecasting financial data: techniques of nonlinear dynamics. Kluwer Academic Publishers: Norwell, Massachusetts
34. Frank RJ, Davey N, Hunt SP (2000) Input window size and neural network predictors. IEEE-INNS-ENNS Int Joint Conf Neural Netw (IJCNN'00)-Vol 2, pp. 2237–2242
35. Merkwirth C, Parlitz U, Wedekind I, Lauterborn W (2002) TSTOOL user manual, http://www.physik3.gwdg.de/tstool/manual.pdf
36. Hagan MT, Demuth HB, Beale MH (1996) Neural network design. PWS Publishing, Boston, MA
37. Hagan MT, Menhaj M (1994) Training feedforward networks with the Marquardt algorithm. IEEE Trans Neural Netw 5(6):989–993
38. Aha D, Kibler DW, Albert MK (1991) Instance-based learning algorithms. Mach Learn 6:37–66
39. Mitchell T (1997) Decision tree learning, machine learning. The McGraw-Hill Companies, Inc., pp 52–78
40. Breiman L, Friedman J, Stone CJ, Olshen RA (1984) Classification and regression trees. Chapman & Hall (Wadsworth, Inc.), New York
41. Stone M (1974) Cross-validatory choice and assessment of statistical prediction. J Roy Statistic Soc B 36:111–120
42. Breiman L (1996) Bagging predictors. Mach Learn 24(2):123–140
43. Dietterich TG (1997) Machine-learning research: Four current direction. AI Magazine 18(4):97–136
44. Hansen L, Salamon P (1990) Neural network ensembles. IEEE Trans Patt Analy Mach Intell (12):993–1001
45. Dietterich TG (2000) Ensemble methods in machine learning. First International Workshop on Multiple Classifier Systems, New York
46. Schapire RE, Freund Y, Bartlett P, Lee WS (1997) Boosting the margin: A new explanation for the effectiveness of voting methods. In: Proceedings of the fourteenth international conference on machine learning. Morgan Kaufmann, pp 322–330
47. Wolpert DH (1992) 'Stacked generalization,' Neural networks. Pergamon Press, vol 5, pp. 241–259
48. Ting KM, Witten IH (1999) Issues in stacked generalization. J Artif Intell Res 10:271–289