Journal of Forecasting
J. Forecast. **29**, 271–284 (2010)
Published online 24 June 2009 in Wiley InterScience
(www.interscience.wiley.com) **DOI**: 10.1002/for.1124

Foreign Exchange Market Prediction with Multiple Classifiers

BO QIAN* AND KHALED RASHEED
Boyd Graduate Studies Research Center, Department of Computer Science, University of Georgia, Athens, Georgia, USA

ABSTRACT
Foreign exchange market prediction is attractive and challenging. According to the efficient market and random walk hypotheses, market prices should follow a random walk pattern and thus should not be predictable with more than about 50% accuracy. In this article, we investigate the predictability of foreign exchange spot rates of the US dollar against the British pound 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, *k*-nearest neighbor, and naïve Bayesian classifier—were then trained with these generated patterns. Through appropriate collaboration of these models, we achieved a prediction accuracy of up to 67%. Copyright © 2009 John Wiley & Sons, Ltd.

KEY WORDS foreign exchange market prediction; efficient market hypothesis; Hurst exponent; machine learning; model ensemble

INTRODUCTION

The foreign exchange market is the largest single market in the world. Its average $1.5 trillion traded per day is about 60 times that of the $25 billion of the New York Stock Exchange. Foreign exchange trading is deep and very liquid. According to the efficient market hypothesis (Fama, 1965; Fama *et al.*, 1969), the current asset market (e.g. stock market, foreign exchange market) price fully reflects all available information. This implies that past and current information are immediately incorporated into the market price; thus price changes are merely due to new information or 'news' rather than existing information. Since news in nature happens randomly and is unpredictable, the best bet for the next price is the current price. If this hypothesis is true, any attempts to predict the market will be fruitless. The efficient market hypothesis is usually tested in the form of a random walk model. The random walk model has been tested extensively in various markets. The results are mixed and sometimes contradictory. In general, many early works (Fama, 1965; Alexander, 1961; Cootner, 1964) support the random walk model. Jensen (1978) claims that 'there is no other proposition in

*Correspondence to: Bo Qian, 415 Boyd Graduate Studies Research Center, Department of Computer Science, University of Georgia, Athens, GA 30602-7404, USA. E-mail: Bo.Qian@rbccm.com economics which has more solid empirical evidence supporting it than the Efficient Market Hypothesis'. However, most recent studies (Butler and Malaikah, 1992; Lo and MacKinlay, 1997; Kavussanos and Dockery, 2001; Gallagher and Taylor, 2002) on financial markets reject the random walk behavior of market prices. In a review, Fama (1991) even states that the efficient market hypothesis surely must be false. Although numerous reports give evidence that prices are not purely random, all agree that the behavior of prices is approximately close to a random walk process. The foreign exchange market is generally considered to be highly efficient, so its prediction is more difficult; it should not be predictable much past 50% (Walczak, 2001). Degrees of accuracy of 56% hit rate in the predictions are often reported as satisfying results (Tsibouris and Zeidenberg, 1995; Baestaens *et al.*, 1996; Giles *et al.*, 1991).

One big problem in financial market prediction is 'self-destruction'. 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' (Hellstrom and Holmstrom, 1998) character of the market, a successful trading strategy must also be dynamic and self-adaptive. Artificial neural networks are nonparametric universal function approximation tools (Hornik *et al.*, 1989) that can learn hidden patterns from the data without assumptions. They are ideal for foreign exchange market prediction. Neural network forecasting models have been widely used in financial time series analysis during the last decade (Refenes, 1995; Gately, 1996; Zirilli, 1997). Given a dataset, even though there might not be an exploitable forecasting relation, we can still get a 'good' model that fits data well by conducting extensive searches. This is called 'data snooping' (White, 2000). 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 three additional inductive machine-learning methods—decision tree, *k*-nearest neighbor and naïve Bayesian classifier—to reduce the risk of data snooping. Through the appropriate ensemble of neural network, decision tree, *k*-nearest neighbor, and naïve Bayesian classifier, we can improve both the accuracy and stableness of our model. This research was conducted using Matlab. All Matlab programs generating results for this paper can be downloaded from http://qianbo.myweb.uga.edu/Research.htm.

The remainder of this article is organized as follows: the next section describes the selection of a period with good predictability by its Hurst exponent. The third section describes the generation of suitable training patterns for classifiers. The fourth section describes four classifiers used in our experiments and various ensemble methods to combine predictions of multiple classifiers. The fifth section then shows our running results. The paper is concluded in the sixth section.

DATA SELECTION

In this study, we examine the US dollar to British pound (USD/GBP) daily exchange rate from January 4 1971 to July 5 2005. The data source is from the Board of Governors of the Federal Reserve System and can be downloaded from http://www.federalreserve.gov/releases/h10/Hist. USD/GBP is one of the most actively traded currency pairs. Numerous papers have discussed forecasting for this pair. However, to the best of our knowledge, no one discusses the selection of a date period for better prediction. The date period is usually chosen arbitrarily without any explanation. In this paper, we use the Hurst exponent to select a period with great predictability for further investigation.

The Hurst exponent, proposed by H. E. Hurst (1951) for use in fractal analysis (Mandelbrot and Ness, 1968; Mandelbrot, 1982) has been applied to many research fields. It has recently become popular in the finance community (May, 1999; Corazza and Malliaris, 2002; Grech and Mazur, 2004) largely due to Peters'(1991, 1994) work. 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 system, it has broad applicability for time series analysis. 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 likely to be 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. Qian and Rasheed (2004) showed that time series with large H are more predictable than those 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 January 4 1971 to July 5 2005. We chose the period length of 1024 days for two reasons. One reason is that it is a good number for the Hurst exponent calculation. The other reason is that it provides enough data for a test, yet it does not suffer much from the time series recency effect (Walczak, 2001).

The time series recency effect is a phenomenon in building time series models that, using data closer in time to the forecast data, usually produces better models. Figure 1 shows the USD/GBP daily return from January 4 1971 to July 5 2004. As is common in the finance community, we used log difference as daily return. Figure 2 shows the corresponding Hurst exponent for this period. In this period, Hurst exponent ranges from 0.4747 to 0.7314. It is interesting to see that in recent years the Hurst exponent is close to 0.5, which is the Hurst exponent value of a random series. This implies that the market has become more efficient in recent years. The largest Hurst exponent value (0.7314) was reached on January 4 1978, so in our experiment we chose the period from December 3 1973 to January 4 1978.

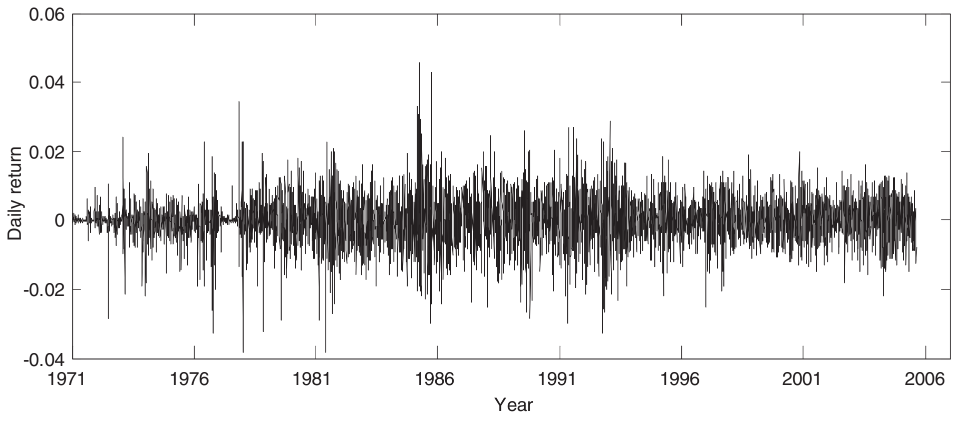

Figure 1. USD/GBP daily returns from January 4 1971 to July 5 2005

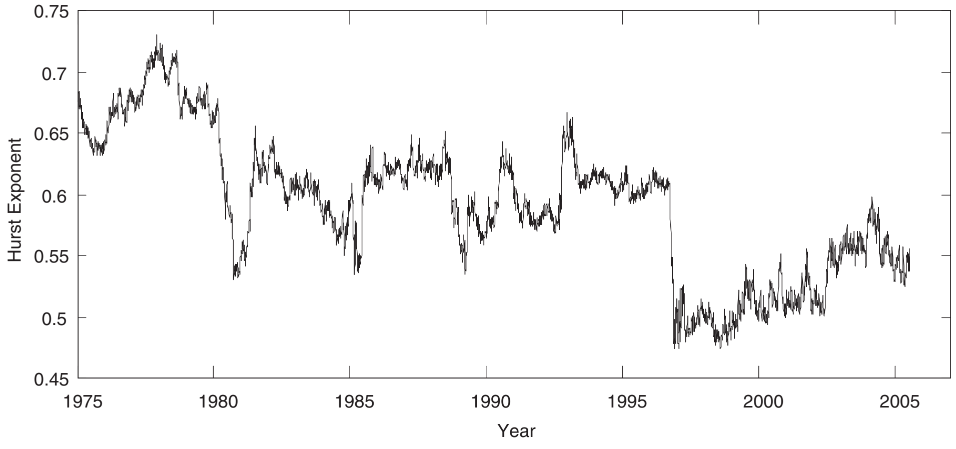

Figure 2. Hurst exponent for USD/GBP daily returns from January 4 1971 to July 5 2005

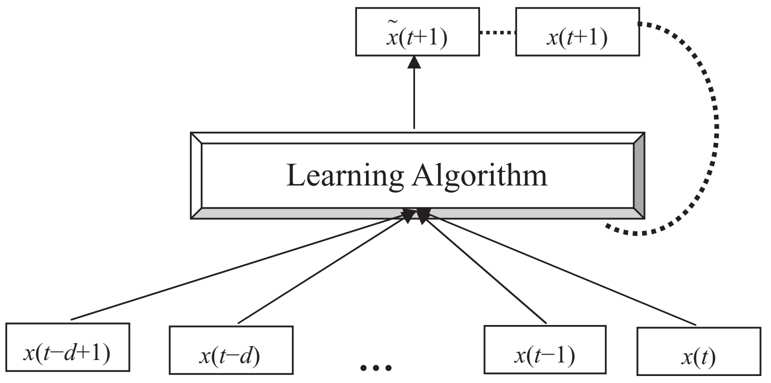

Figure 3. Sliding window training pattern for a learning algorithm performing time series prediction

TRAINING PATTERN GENERATION

Now that we had USD/GBP daily return time series from December 3 1973 to January 4 1978, we want to predict the return sign (up or down direction) based on previous returns. An obvious question is: how do we generate training patterns to be used in machine learning 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 $\mathbf{X}_1 = (x_1, x_2, x_3, \ldots, x_d)$, $\mathbf{X}_2 = (x_2, x_3, x_4, \ldots, x_{d+1}) \ldots$, etc. are generated. Each input vector makes up one training pattern for unsupervised machine learning; paired with a target output value, it makes up a training pattern for supervised learning. Figure 3 gives a basic architecture for such a learning model performing time series prediction.

If the learning algorithm is a linear combination of the inputs, this can be seen as an auto-regressive time series modeling. For multiple-layer neural networks with a nonlinear transfer function, acting as universal function approximators, the technique essentially gives a function f, which maps a d-dimensional input space R^d to an output space R.

However, input vectors can also be $\mathbf{X}_1 = (x_1, x_3, x_5, \ldots, x_{2d-1})$, $\mathbf{X}_2 = (x_2, x_4, x_6, \ldots, x_{2d}) \ldots$, etc. We need to choose a vector size d and separation τ (1 day, 2 days, etc.). Since more and more evidence (Peters, 1991, 1994; Hsieh, 1991) shows that financial markets are chaotic nonlinear dynamic systems, we use heuristics from chaos theory to determine d and τ. Taken's (1981) theorem 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 time-delay or separation. Using the auto-mutual information and Cao's false nearest-neighbor methods (Cao, 1997; Soofi and Cao, 2002), we can estimate d and τ. Frank *et al.* (2000) have shown that the estimated d and τ by these methods are good heuristics to generate training patterns for prediction. We used the TSTOOL package (Merkwirth *et al.*, 2002) to run the auto-mutual information and false nearest-neighbor methods for our data. Figure 4 gives the auto-mutual information for USD/GBP daily return from December 4 1973 to January 3 1978. From this figure, the first minimum of auto-mutual information is at 1, so we chose separation $\tau = 1$. Figure 5 gives the result of Cao's false nearest-neighbor method for minimum embedding dimension estimation. E1(d) measures 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 the value d at which the increment of E1(d) slows down and begins to saturate. There is a kink in the graph at dimension 6, so we chose embedding dimension $d = 6$.

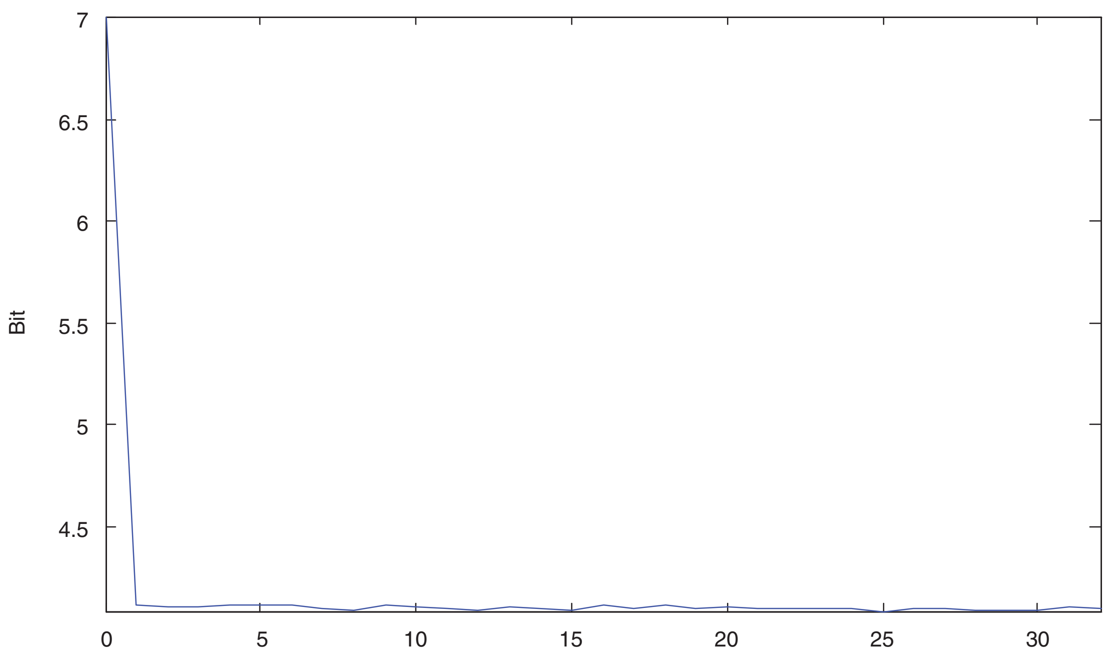

Figure 4. Auto-mutual information for USD/GBP daily returns from December 3 1973 to January 4 1978

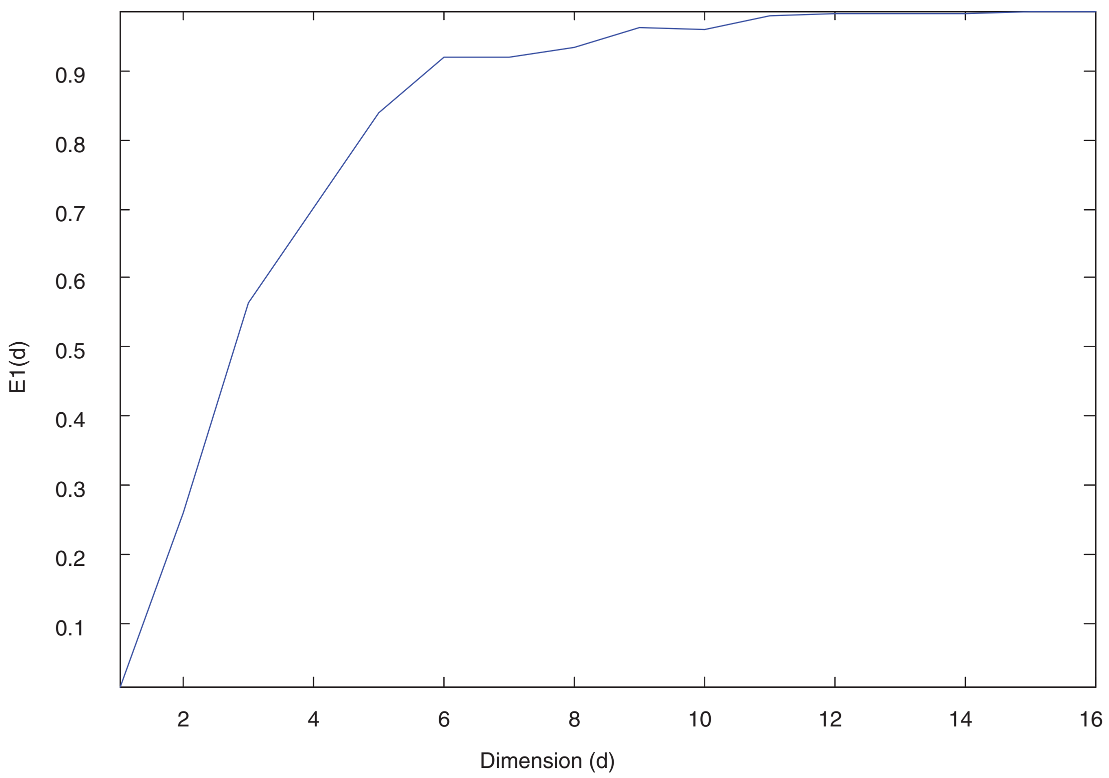

Figure 5. Minimum embedding dimension for USD/GBP daily returns from December 3 1973 to January 4 1978 using Cao's method

Since $\tau = 1$ and $d = 6$ are suggested, we used every set of six continuous daily returns to predict the direction of the seventh day. Thus, 1018 training patterns were generated, each of the form (r_{i-5} r_{i-4} r_{i-3} r_{i-2} r_{i-1} r_i; d_{i+1}), where r_i was the daily return in day i and d_{i+1} was the moving direction, which is 1 if $r_{i+1} > 0$ and -1 if $r_{i+1} < 0$. Since all inputs r_{i-5}, r_{i-4}, r_{i-3}, r_{i-2}, r_{i-1} and r_i are in the same scale, it was not necessary to do any normalization. These 1018 patterns formed the whole dataset for our machine learning classifiers. We used the last 20% (204 patterns) of the data as out-of-sample data to judge the performance of the final models. This out-of-sample data were never used in any model generation or evaluation processes. The remaining 80% of the data (814 patterns) were used as in-sample data to develop models.

CLASSIFIERS

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

Artificial neural network

We used a feed-forward single-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 (Merkwirth *et al.*, 2002) 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 back-propagation technique (Hagan and Menhaj, 1994) that is much less complex than computing the Hessian matrix. The Levenberg–Marquardt algorithm usually converges most quickly and with the lowest mean squared 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

k-Nearest neighbor (Aha *et al.*, 1991) is an instance-based learning algorithm. It uses similar cases to predict unknown cases. In our experiments, 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 case of a tie, we break the tie according to the average distance.

Decision tree

Decision trees (Mitchell, 1997) 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 a range of values (for a continuous attribute)**. In our experiments, the CART (Breiman *et al.*, 1984) 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.

Naïve Bayesian classifier

The naive Bayesian classifier (Nir *et al.*, 1997) technique is based on the so-called Bayesian theorem. A naive Bayesian classifier is, in essence, a model of a joint probability distribution over a set of feature variables. By assuming that each feature is conditionally independent from the other features and through Bayes' rule, the posterior joint probability is decomposed to a series of simple prior probability values. Given an instance as a combination of values for the feature variables, the instance is classified as the class with the largest posterior probability. The word 'naïve' comes from the conditionally independent assumption. Although the assumption is not always true, naive Bayes can often outperform more sophisticated classification methods (Rish, 2001). In our experiments, we used a normal distribution density function to calculate the prior probability for each feature variable.

Parameter determination

Aside from the settings described above, for every learning algorithm except the naïve Bayesian classifier, there was still one major parameter to be determined. 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 *k* values from

1 to 10. Since the prediction results of these classifiers are sensitive to data split, we used cross-validation (Stone, 1974) to judge their generalization ability. Because we used 20% of the data as a test dataset, to simulate this partition we used a fivefold cross-validation method to evaluate models. For the *k*-nearest neighbor, the remaining 80% of the data were used as training data. For the neural network and the decision tree, 20% of the data were used as a validation set, and the other 60% as a training set. The validation dataset 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 (Breiman, 1996). For each classifier, we ran a fivefold cross-validation 10 times. Minimum, maximum and average classification error rates for each parameter setting were recorded. Tables I–III show the fivefold cross-validation results for the neural network, *k*-nearest neighbor and decision tree. For each classifier we chose the parameter value with minimum average cross-validation error rate for our subsequent experiments. The minimum error rate is marked in bold in each table.

Classifier ensembles

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 (Dietrich and Winter, 1997). An ensemble of classifiers is a set of classifiers

Table I. Error rates (%) of neural network cross-validation for 1–10 hidden nodes

	1	2	3	4	5	6	**7**	8	9	10
Minimum	43.62	44.70	45.59	43.99	44.84	44.83	45.20	45.68	45.32	44.47
Maximum	49.77	49.76	49.03	48.03	49.01	48.92	47.66	48.68	48.78	49.05
Average	46.81	46.91	47.42	46.25	47.28	46.65	**45.98**	46.95	46.41	46.73
SD	1.86	1.63	1.19	1.26	1.46	1.25	0.79	1.01	0.97	1.49

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

	1	2	3	4	5	6	**7**	8	9	10
Minimum	43.95	44.60	43.86	45.19	45.08	44.98	43.97	44.60	43.38	44.45
Maximum	46.81	48.28	48.79	46.92	47.81	47.66	46.93	47.55	48.52	47.81
Average	45.77	46.67	45.96	45.80	45.57	46.32	**45.35**	46.07	45.96	46.21
SD	1.05	1.16	1.51	0.59	0.83	0.83	1.14	1.01	1.31	0.90

Table III. Error rates (%) of decision tree cross-validation for 1–10 maximum impure cases in a leaf

	1	2	3	4	5	6	7	**8**	9	10
Minimum	45.83	45.09	44.22	43.63	43.72	44.57	43.99	43.13	44.12	44.83
Maximum	48.14	47.42	47.28	47.54	47.66	48.76	47.53	47.55	47.18	47.66
Average	46.94	46.15	45.92	45.73	45.49	46.38	46.14	**45.48**	46.19	45.81
SD	0.82	0.73	0.95	1.10	1.48	1.37	1.25	1.46	0.88	0.86

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 individual classifier, the classifiers must be accurate and diverse (Hansen and Salamon, 1990). Here, 'accurate' means better than random guessing and 'diverse' means the classifiers make uncorrelated errors. Research has shown that ensembles are often more accurate in practice than the best of the classifiers they are composed of (Dietterich, 2000). Since the accuracy of financial market prediction is usually a 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 which can continue to function even if one underlying classifier fails. This consistency is a very important character of successful systems for investment decisions.

Models that have been derived from different executions of the same learning algorithm are usually called homogeneous. Bagging (Breiman, 1996) and boosting (Schapire *et al.*, 1997) are 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 dataset are called heterogeneous. Voting and stacking (Wolpert, 1992) are two effective methods to combine the results of heterogeneous classifiers. In our experiment, we investigated the simplest unweighted voting and the state-of-art stacking methods. In unweighted voting, each model outputs a class value and has one vote for the class, and then the output by 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 the outputs 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 dataset 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 (1992) is elaborated as follows.

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

1. Generate the training dataset for the meta-learner
   For each classifier C_j $(1 \le j \le J)$, use $L^{(-k)}$ $(1 \le k \le K)$ as training data to build model $M_j^{(-k)}$. Then use the remaining data $L^k(\mathbf{x})$ as input for model $M_j^{(-k)}$ and get output $L_j^k(\tilde{y})$. After training by all K folds, we get output $L_j(\tilde{y}) = \{L_j^k(\tilde{y}) \mid 1 \le k \le K\}$, 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 dataset LM for the meta-learner:

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

2. Build model for the meta-learner
   In this step, use LM as the 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 ($1 \le j \le J$) for each classifier.
4. Prediction
   To classify a new instance $\mathbf{x}$, we input the vector $\mathbf{x}$ to models M_j ($1 \le j \le J$). 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 $\mathbf{x}$. This completes the procedures proposed by Wolpert.

Ting and Witten (1999) 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 $\mathbf{x}_i$, model $M_j^{(-k)}$ outputs a vector $\mathbf{p}_{ji} = (p_1, p_2, \ldots, p_T)$, the predicted probabilities of $\mathbf{x}_i$ belonging to each class. In this case:

$$\mathrm{LM} = \{(\mathbf{p}_{1i}, \mathbf{p}_{2i}, \ldots, \mathbf{p}_{Ji}; y_i), 1 \le i \le I\}$$

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

We notice 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 dataset $(\tilde{y}_{1i}, \tilde{y}_{2i}, \ldots, \tilde{y}_{Ji})$ 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 work, 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 $f(M, \mathbf{x})$ be the output of model M given input $\mathbf{x}$. M_j' is defined as

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

That is, the output of M_j' is the average of models $M_j^{(-k)}$ ($1 \le k \le K$). We think model M_j' is a better representation than M_j for the training dataset generated by models $M_j^{(-k)}$. Note 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 also 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. In the remainder of this article, we refer to Wolpert and Ting's stacking model as SM and our stacking model as SM′.

EXPERIMENTAL RESULTS

Results for individual classifiers

Using the configuration and parameters discussed above, we ran each classifier 30 times. As we mentioned previously, we used the last 20% of the data as a prediction dataset to judge the final performance. Of the remaining data, the last 20% were used as test data. For each run, we executed each learning algorithm five times to produce five models. The model with the smallest error rate for the test data was selected as model M_j. The prediction data were then input to this model, and its output was used as the prediction result. Error rates for both test and prediction data were recorded and the mean and standard deviation of 10 runs were calculated. For fivefold cross-validation, we produced one model $M_j^{(-k)}$ for each fold. The prediction data was then input to these five models.

The average error rate of these five models was calculated as a performance measure for the fivefold prediction. The average prediction of the five models was generated as the output of model M'_j and the error rate of this fivefold average prediction was recorded. Meanwhile, the combination of outputs of the fivefold test data for model $M_j^{(-k)}$ was generated as meta-learning training data for stacking purposes. Table IV shows the test and prediction error rates for each classifier.

From Table IV we can see that although k-nearest neighbour had the smallest error rates for the test data, the smallest error rates for the prediction data were achieved by the neural network. This means the simple model evaluation and selection method does not work well in this case. It provides an example for the need to combine multiple classifiers. We also notice that the fivefold average prediction model M'_j achieved the best performance among M_j, $M_j^{(-k)}$ and M'_j. This seems reasonable, as M'_j is in essence an ensemble of five $M_j^{(-k)}$ models.

Results for ensembles of classifiers

Table V shows the results for stacking and voting. From Table V we can see that our fivefold average prediction method was much better than the standard stacking method. This confirms our thought that M'_j is a better representation than M_j. Also the performance of simple unweighted voting was almost the same as that of the state-of-the-art stacking method in our experiments. This suggests that a more complex model is not necessarily a better one. To our surprise, the performance of these ensembles did not exceed that of the best individual classifier. One possible reason for this kind of performance is the poor prediction by the decision tree. The other possibility is that the predictions of the classifiers were not independent, but instead highly correlated. To test this possibility, we investigated how many predictions were agreed upon by all four classifiers. There were a total of 6120 (204 instances in prediction set $\times$ 30 runs) predictions made. Assuming 55% accurate prediction rate, in theory two classifiers will agree on 50.5% ($0.55^2 + 0.45^2$) of the instances, three classifiers will agree on 25.75% ($0.55^3 + 0.45^3$), and four classifiers will agree on 13.25% ($0.55^4 + 0.45^4$) if their predictions are independent. This gives theoretical consensus numbers of 3091 for two classifiers, 1576 for three classifiers, and 811 for four classifiers. Table VI gives the number of consensus predictions for the different classifiers in our experiments. We can see that the numbers were much higher than their theoretical counterparts. This indicates a high correlation of predictions between

Table IV. Error rates for neural network, k-nearest neighbor, decision tree and naïve Bayesian classifier

	Neural network	k-Nearest neighbor	Decision tree	Naïve Bayesian	Avg. of classifiers
Test (M_j)	46.42	**46.30**	46.62	47.18	46.63
Prediction (M_j)	**45.15**	46.82	49.38	45.82	46.79
5-fold test ($M_j^{(-k)}$)	46.17	**44.77**	45.54	47.18	45.92
5-fold prediction ($M_j^{(-k)}$)	**44.89**	45.52	47.96	47.93	46.58
5-fold average prediction (M'_j)	**43.09**	43.22	46.59	47.65	**45.14**

Table V. Error rates for different ensemble methods

	Stacking	Voting	Consistent Voting
Prediction output (M_j)	46.51 (SM)	46.01	40.99
5-fold average prediction output (M'_j)	**43.23 (SM′)**	43.37	**33.30**

Table VI. Number of consistence instances and error rates of consistent voting

	ANN-kNN # (error rate)	ANN-DT # (error rate)	ANN-NBC # (error rate)	kNN-DT # (error rate)	kNN-NBC # (error rate)	DT-NBC # (error rate)
Prediction output	3436 (42.84%)	3015 (44.44%)	3649 (42.42%)	3315 (46.49%)	3267 (43.1%)	3066 (45.21%)
5-fold average prediction output	3625 (37.60%)	3680 (41.41%)	2881 (40.12%)	3629 (40.56%)	3006 (39.65%)	2885 (43.85%)

	ANN-kNN-DT # (error rate)	ANN-kNN-NBC # (error rate)	ANN-DT-NBC # (error rate)	kNN-DT-NBC # (error rate)	ANN-kNN-DT-NBC # (error rate)
Prediction output	1823 (42.68%)	2116 (40.74%)	1805 (41.55%)	1764 (43.31%)	1149 (40.99%)
5-fold average prediction output	2407 (36.52%)	1696 (34.73%)	1663 (38.80%)	1700 (37.76%)	**1075 (33.30%)**

classifiers. In this article, in addition to ensemble methods we studied the performance of a consistent voting method—a method that only counts predictions agreed upon by all classifiers. This is appropriate for our foreign exchange market prediction application because we can simply take no trading action if the prediction is not unanimously agreed upon by all classifiers. Table VI shows the error rates for consistent voting of two, three and four 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. When all four classifiers agreed in fivefold average prediction, we achieved a 33.30% error rate. This is a tremendous improvement over random guessing.

CONCLUSION

In this article we attempted to predict the USD/GBP daily return direction. Using the Hurst exponent and heuristics from chaos theory, we achieved about 55% accuracy for individual classifiers. Since the Hurst exponent provides a measure for predictability, we could use this value to guide data selection before forecasting. This saved time and led to superior results. In addition, we further investigated ensemble methods to improve prediction performance. Simple voting and stacking ensemble methods did not work well due to one poorly performing classifier and high correlations between classifiers. However, even in this situation a consistent voting ensemble method could increase the prediction accuracy from 57% to 67%.

REFERENCES

Aha DW, Kibler D, Albert MK. 1991. Instance-based learning algorithms. *Machine Learning* **6**: 37–66.
Alexander SS. 1961. Price movements in speculative markets: trends or random walks. *Industrial Management Review* May: 7–26.

Baestaens DE, van den Bergh WM, Vaudrey H. 1996. Market inefficiencies, technical trading and neural networks. In *Forecasting Financial Markets, Financial Economics and Quantitative Analysis*, Christian D (ed.). Wiley: Chichester; 254–260.

Breiman L. 1996. Bagging predictors. *Machine Learning* **24**(2): 123–140.

Breiman L, Friedman JH, Stone CJ, Olshen RA. 1984. *Classification and Regression Trees.* Chapman & Hall (Wadsworth, Inc.): New York.

Butler KC, Malaikah JS. 1992. Efficiency and inefficiency in thinly traded stock markets: Kuwait and Saudi Arabia. *Journal of Banking and Finance* **16**(1): 197–210.

Cao L. 1997. Practical method for determining the minimum embedding dimension of a scalar time series. *Physica D* **110**: 43–50.

Cootner PH. 1964. *The Random Character of Stock Market Prices.* MIT Press: Cambridge, MA.

Corazza M, Malliaris AG. 2002. Multi-fractality in foreign currency markets. *Multinational Finance Journal* **6**(2): 65–98.

Dietterich TG. 2000. Ensemble methods in machine learning. In *First International Workshop on Multiple Classifier Systems*, New York.

Dietterich TG, Winter 1997. Machine-learning research: four current direction. *AI Magazine* **18**(4): 97–136.

Fama EF, Fisher L, Jensen MC, Roll WR. 1969. The adjustment of stock price to new information. *International Economic Review* **10**(1): 1–21.

Fama EF. 1991. Efficient capital markets: II. *Journal of Finance* **46**(5): 1575–1617.

Fama EF. 1965. The behaviour of stock market prices. *Journal of Business* **38**: 34–105.

Frank RJ, Davey N, Hunt SP. 2000. Input window size and neural network predictors. In *IEEE-INNS-ENNS International Joint Conference on Neural Networks (IJCNN'00)*, Vol. 2; 2237–2242.

Gallagher LA, Taylor MP. 2002. Permanent and temporary components of stock prices: evidence from assessing macroeconomic stocks. *Southern Economic Journal* **69**: 245–262.

Gately E. 1996. *Neural Networks for Financial Forecasting.* Wiley: New York.

Giles CL, Lawrence S, Tsoi AC. July 2001. Noisy time series prediction using a recurrent neural network and grammatical inference. *Machine learning* **44**(1–2): 161–183.

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

Hagan MT, Menhaj M. 1994. Training feedforward networks with the Marquardt algorithm. *IEEE Transactions on Neural Networks* **5**(6): 989–993.

Hagan MT, Demuth HB, Beale MH. 1996. *Neural Network Design*. PWS Publishing: Boston, MA.

Hansen LK, Salamon P. 1990. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence* **12**: 993–1001.

Hellstrom T, Holmstrom K. 1998. *Predicting the Stock Market*. Technical Report Series IMa-TOM-1997-07. Center of Mathematical Modeling, Mälardalen University, Västerås, Sweden.

Hornik K, Stinchcombe MB, White H. 1989. Multilayer feedforward networks are universal approximators. *Neural networks* **2**(5): 259–366.

Hsieh DA. 1991. Chaos and nonlinear dynamics: application to financial markets. *Journal of Finance* **46**: 1839–1877.

Hurst EH. 1951. Long-term storage of reservoirs: an experimental study. *Transactions of the American Society of Civil Engineers* **116**: 770–799.

Jensen MC. 1978 Some anomalous evidence regarding market efficiency. *Journal of Financial Economics* **6**: 95–102.

Kavussanos MG, Dockery E. October 2001. A multivariate test for stock market efficiency: the case of ASE. *Applied Financial Economics* **11**(5): 573–579.

Lo AW, MacKinlay CA. 1997. Stock market prices do not follow random walks. *Market Efficiency: Stock Market Behaviour in Theory and Practice* **1**: 363–389.

Mandelbrot BB. 1982. *The Fractal Geometry of Nature.* W. H. Freeman: New York.

Mandelbrot BB, Ness WVJ. 1968. Fractional Brownian motions, fractional noises and applications. *SIAM Review* **10**: 422–437.

May CT. 1999. *Nonlinear Pricing: Theory and Applications.* Wiley: New York.

Merkwirth C, Parlitz U, Wedekind I, Lauterborn W. 2002. TSTOOL user manual. http://www.physik3.gwdg.de/tstool/manual.pdf [5 February 2006].

Mitchell T. 1997. Decision tree learning. In *Machine Learning*, Mitchell T (ed.). McGraw-Hill: New York; 52–78.

Nir F, Dan G, Moises G. 1997. Bayesian network classifier. *Machine Learning* **29**: 131–163.

Peters EE. 1991. *Chaos and Order in the Capital Markets: A New View of Cycles, Prices, and Market Volatility.* Wiley: New York.

Peters EE. 1994. *Fractal Market Analysis: Applying Chaos Theory to Investment and Economics*. Wiley: New York.

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; 203–209.

Refenes AP. 1995. *Neural Networks in the Capital Markets*. Wiley: New York.

Rish I. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence*, Seattle, WA.

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: San Francisco, CA; 322–330.

Soofi AS, Cao L. 2002. *Modelling and Forecasting Financial Data: Techniques of Nonlinear Dynamics*. Kluwer: Norwell, MA.

Stone M. 1974. Cross-validatory choice and assessment of statistical prediction. *Journal of the Royal Statistical Society B* **36**: 111–120.

Takens F. 1981. *Dynamical System and Turbulence*. Lecture Notes in Mathematics 898. Springer: Berlin.

Ting K, Witten IH. 1999. Issues in stacked generalization. *Journal of Artificial Intelligence Research* **10**: 271–289.

Tsibouris G, Zeidenberg M. 1995. Testing the efficient markets hypothesis with gradient descent algorithms. In *Neural Networks in the Capital Markets*, Refenes AP (ed.). Wiley: Chichester; 127–136.

Walczak S. 2001. An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of Management Information Systems* **17**(4): 203–222.

White H. 2000. A reality check for data snooping. *Econometrica* **68**(5): 1097–1126.

Wolpert DH. 1992. Stacked generalization. *Neural Networks* **5**: 241–259.

Zirilli JS. 1997. *Financial Prediction using Neural Networks.* International Thomson Computer Press: London.

Authors' biographies:

Bo Qian obtained his PhD in computer science from University of Georgia in 2006. His research focuses on financial market prediction and machine learning models.

Khaled Rasheed obtained his PhD in Computer Science from Rutgers, the State University of New Jersey in 1998. He is currently an Associate Professor in the Computer Science Department at the University of Georgia (UGA), Athens, Georgia, USA. He is also the graduate Coordinator of the UGA Institute for Artificial Intelligence and a member of the UGA Institute of Bioinformatics and the UGA faculty of Engineering. His current research interests are in the areas of Bioinformatics, Evolutionary Computation, and Machine Learning.

Authors' addresses:
Bo Qian and **Khaled Rasheed**, 415 Boyd Graduate Studies Research Center, Department of Computer Science, University of Georgia, Athens, GA 30602–7404, USA.