# Using machine learning algorithms to find patterns in stock prices*

Pedro N. Rodriguez
*Universidad Complutense de Madrid*

Simon Sosvilla-Rivero
*FEDEA* and *Universidad Complutense de Madrid*

First draft: March 6, 2006
This draft: October 25, 2006

## Abstract

We use a machine learning algorithm called *Adaboost* to find direction-of-change patterns for the S&P 500 index using daily prices from 1962 to 2004. The patterns are able to identify periods to take long and short positions in the index. This result, however, can largely be explained by first-order serial correlation in stock index returns.

JEL Classification Numbers: C45, G11, G14

Keywords: Direction-of-change predictability, Machine learning algorithms, Adaboost

*Pedro N. Rodriguez thanks CONACYT (Mexico) for financial support (Fellowship: 170328). Address correspondence to Professor Sosvilla-Rivero, Fundación de Estudios de Economía Aplicada (FEDEA), C/ Jorge Juan, 46, 28001 – Madrid, Spain, phone: +34 914350401, fax: +34 915779575 e-mail: simon.sosvilla@fedea.es.

## 1. – Introduction

Is a move upward or downward in stock prices predictable? A considerable amount of work has been devoted to examining whether or not this is feasible. Even though the presence of linear predictable components in stock returns is nowadays widely accepted (see, e.g., Fama [1], Lo and MacKinlay [2], Conrad and Kaul [3], Jegadeesh [4] and Kaul [5]), the existence of a function (or formula) which expresses the likelihood of a market fluctuation is not.

Recent advances in both analytic and computational methods, however, have helped empirical investigation on the behavior of security prices. Particularly, direction-of-change (or sign) predictability is currently evaluated via either supervised learning techniques or machine learning algorithms or classifier induction techniques (see, e.g., Apte and Hong [6], Tsaih, Hsu, and Lai [7], Zemke [8], Chen, Leung, and Daouk [9], Kim [10], Rodriguez and Rodriguez [11] and O'Connor and Madden [12], among others). Although this branch of research provides evidence in support of the existence of a function that discriminates up from down movements, it is not clear whether or not machine learning algorithms are extracting information beyond that contained in autocorrelation patterns.

In this paper, we reexamine the sample evidence of direction-of-change predictability in weak-form tests. In particular, we use a machine learning algorithm that is among the most popular and most successful for classification tasks called *Adaboost*. One of the main properties that make the application of Adaboost to financial databases

interesting is that it has showed, in many applications (albeit in non-financial databases), robustness against overcapacity and produced, in many cases, low test-set error.

When we apply Adaboost to S&P 500 daily data, one main conclusion emerges about stock return predictability. We show that periods characterized by high first-order serial correlation in stock returns allow both in-sample and out-of-sample direction-of-change predictability. In essence, the lack of autocorrelation in stock returns does not permit Adaboost to discover a function that discriminates between upwards and downwards movements better than random. Indeed, simple random classifiers (i.e., coin-toss classifiers) are able to explain the apparent predictability in such periods.

In Section 2, we provide a brief review of machine learning algorithms and describe in detail the specific machine learning algorithm we use in our analysis: Adaboost. We apply this algorithm to the daily returns of the S&P 500 stock index from 1962 to 2004 and report the results in Section 3. To check the accuracy of our predictions, we estimate several random classifiers and autoregressive models and the results are also given in Section 3. Finally, in Section 4 we offer some concluding remarks.

## 2. - Machine Learning Algorithms and Adaboost

The starting point for any study of stock return predictability is the recognition that prices, or more specifically, returns develop in either a linear or nonlinear fashion

over time and that their behavior contain certain stable patterns.[1] In order to obtain those patterns, we start by declaring that stock price movements $\{y\}$ satisfy an expression like the following:

$$y_t = \varphi(x_{t-1,z,j},...,x_{t-1,F,j}) + \varepsilon_t \quad (1)$$

where $F$ is the number of potential predictors (or inputs), $x_{t-1,z,j}$ is the realization of the factor $z$ for the asset $j$ at time $t-1$, $\varphi(\bullet)$ is an unknown smooth function that maps the lagged predictors to the response variable, $\varepsilon_t$ is the noise component and $y$ is the "output" or "response" variable $y \in C$, where $C$ is the set of class labels. In this paper, $y_t \to 1$ if$(R_{t,j} - R_t^f > 0)$, 0 otherwise.[2] In other words, positive equity premiums were codified with 1's. Hence, we consider here a two-class case, i.e., $y_t \in C = \{0,1\}$.

When stock price movements are expressed as in Equation (3), it is evident that quantitative patterns may emerge from the application of machine learning algorithms. But just how useful are these uncovered patterns?

To answer this question empirically, we must test the in-sample and, more importantly, the out-of-sample discriminatory accuracy of the learning algorithms used to uncover $\hat{\varphi}(\cdot)$. In Section 2.A, we provide a brief review of Adaboost. Section 2.B briefly describes tree-based models.

[1] As pointed out by Bossaerts and Hillion [13], there are not intuitive economic reasons why the set of significant variables should erratically change form one month to the next.

[2] $R_{t,j}$ is the return of asset $j$ at time $t$ and $R_t^f$ is the risk-free return at time $t$.

*A. Adaboost*

Boosting was created from the desire to transform a collection of weak classifiers into a strong ensemble or weighted committee. It is a general method for improving the performance of any learning algorithm. Boosting was proposed in the computational learning theory literature by Schapire [14] and Freund [15]. Freund and Schapire [16] solved many practical difficulties of earlier boosting algorithms with the creation of Adaboost.

Much has been written about the success of Adaboost in producing accurate classifiers. In fact, one of the main characteristic of this procedure is that the test error seems to consistently decrease and then level off as more classifiers are added, without having an ultimately increase. The main steps of the Adaboost's algorithm are:[3]

1. Start with weights $w_i = 1/n, i = 1, ..., n$
2. For $m = 1, \ldots, M$ do:
   (a) Fit the machine learning algorithm $\hat{\varphi}_m(\cdot)$ using weights $w_i$ on the training data.
   (b) Compute $\text{err}_\text{m} = E_w[I_{(y \neq \hat{\varphi}_m(\cdot))}]$ and $c_m = \log((1 - \text{err}_\text{m})/\text{err}_\text{m})$.
   (c) Update $w_i \leftarrow w_i \exp[c_m \cdot I_{(y_i \neq \hat{\varphi}_m(\cdot))}], i = 1, ..., n$ and renormalize so that $\sum w_i = 1$.
3. Output $\hat{\varphi}(\cdot) = \text{sign}[\sum_{m=1}^{M} c_m \hat{\varphi}_m(\cdot)]$.

[3] $I_{(S)}$ is the indicator function of the set $S$ and $E_w$ denotes the weighted average of error with weights $w = (w_1, ..., w_n)$.

Note that at step *m*, those observations that were misclassified by the classifier $\hat{\varphi}_{m-1}(\cdot)$ induced at the previous step have their weights increased; therefore, as Hastie *et al.* [17, 300-301] clearly state "[e]ach successive classifier is thereby forced to concentrate on those training observations that are missed by previous one in the sequence."

In this paper, we use 2-node tree-based models, also known as stumps, as base learners: the machine learning algorithm used to obtain $\hat{\varphi}_m(\cdot)$ at each iteration. The implementation was carried out in R: Environment for Statistical Computing and Graphics with the *ada* add-on package developed by Culp, Johnson, and Michailidis [18].

*B. Tree-based models*

The origins of classification trees or hierarchical classification come from two areas of investigation. In the field of statistical pattern recognition Breiman, Friedman, Olshen, and Stone [19] developed a technique named CART (Classification and Regression Trees). The Machine Learning community provided a computer program called ID3, which evolved into a new system named C4.5 (see Quinlan [20,21]).

Tree-based techniques involve partitioning the explanatory variables space into a set of rectangles and then fit a simple model to each one. A tree-based model tries to find the split that maximizes the decrement in a loss function in order to make a tree grow. This is done iteratively until a certain amount of observations is reached or no further decrements in the loss function are found. More formally, a tree may be expressed as,

$$T(\mathbf{x};\Theta) = \sum_{j=1}^{J} \gamma_j I(\mathbf{x} \in R_j), \tag{2}$$

with parameters $\Theta = \{R_j, \gamma_j\}_1^J$. Where $\gamma_j$ (a constant) is assigned to a region ($R_j$). The constant can be a value, a probability or a class label assigned to an element in the region $R_j$. $J$ is usually treated as a meta-parameter and can be interpreted as the maximum amount of admissible interactions among explanatory variables less one, and $I(\bullet)$ is an indicator function. It is worth mentioning that *J* also represents the stopping criteria of the top-down algorithm of the tree-based models (briefly described below) and that we fixed to two. In other words, we use the so-called stumps. Here the parameters $\Theta = \{R_j, \gamma_j\}_1^J$ are found by minimizing the empirical risk, like in the following equation:

$$\hat{\Theta} = \arg\min_{\Theta} \sum_{j=1}^{J} \sum_{\mathbf{x}_i \in R_j} L(y_i, \gamma_j) \tag{3}$$

where $L(\bullet)$ denotes a loss function. This is an extraordinary combinatorial optimization problem, so we must rely on sub-optimal solutions. The aforementioned optimization problem can be divided into two parts. The first one, finding $\gamma_j$ given $R_j$, is typically trivial, where $\hat{\gamma}_j$ is the modal class of observations falling in region $R_j$. The difficulty of this combinatorial optimization problem is based on finding $R_j$. A helpful solution is to employ heuristic methods.

Safavian and Landgrebe [22] provide a survey on heuristic methods proposed for designing decision trees. The most popular heuristic method in tree-based models is the top-down recursive partitioning, which starts with a single region covering the entire

space of all joint input values. This is partitioned into two regions by choosing an optimal splitting input variable $x_j$ and a corresponding optimal split point *s*. Values in **x** for which $x_j \le s$ are defined to be the left daughter region, and those for which $x_j > s$ denote the right daughter region. Then each of these two daughter regions is optimally partitioned with the same strategy, and so forth.

In this paper, we replaced the loss function with the Gini index, given by

$$\text{Gini index}: \sum_{k=1}^{K} \hat{p}_{mk}(1-\hat{p}_{mk}). \tag{4}$$

where in a node *m*, representing a region $R_j$, let $\hat{p}_{mk}$ be the proportion of class *k* observations in the node *m*, and *K* represents the total number of classes or populations under study. The implementation was carried out in R: Environment for Statistical Computing and Graphics with the *ada* add-on package, developed by Culp, Johnson, and Michailidis [18].

## 3. – Empirical Results

The empirical application uses S&P 500 daily closing prices from August 7, 1962 to December 31, 2004. The data set was divided into four non-overlapping sets. Since it is well-known that the ultimate measure of quality of a learner is its generalization performance, we divided each set into two sub-samples. The first sub-sample is used for training, whereas the second sub-sample is used for testing. We assume that future stock price movements $\{y\}$ may be related to past returns, as in the following equation:

$$y_i = f(r_{i-1},...,r_{i-24}), i = 25,...,n \tag{5}$$

where $y_i$ equals 1 if the return observed at time $i$ $\{r_i\}$ is greater than zero, 0 otherwise. We applied the algorithm described in Section 2.A to the each training set. The results are shown in Table 1.

**********
Table 1 about here
**********

Table 1 displays several accuracy measures, as explained below. We use these accuracy measures to analyze the properties of Adaboost's in-sample and out-of-sample performance. We study six specifications corresponding to iterations {*M*} equal to 1, 2, 25, 50, 200, and 1000.

The in-sample and out-of-sample accuracy measures are *(a)* the error rate, which corresponds to the total number of misclassified observations divided by the total number of observations; *(b)* the bias, defined as systematic loss incurred by the function; *(c)* the unbiased variance (denoted by *Vu*) evaluates the extent to which the estimated function deviates from the correct predictions; and *(d)* the biased variance (denoted as *Vb*) assesses the extent to which the estimated function deviates from the incorrect predictions. These accuracy measures are described in more technical detail in the Appendix.

In the first two data sets, the bias plays a significant role in its contribution to the error rate. In other words, the systematic loss incurred by the functions is higher than the total error rate. Moreover, Adaboost's error has a positive relationship with the total

number of iterations $\{M\}$. Evidently, this later result indicates that Adaboost rapidly over-fits the data.

In contrast, in the last two data sets, the bias is lower than the error rate. This only occurs when the loss incurred by function's fluctuations around the central tendency in response to different samples has a direct effect on error. Note also how the error decrease has the number of iterations increases.

We simulated 1000 coin-toss classifiers for each data set to analyze the extent to which the results reported in Table 1 can be explained by randomness. To obtain each random classifier, we generate random values from a discrete distribution in which two values where possible: 1's and 0's. Each value was assigned 50 per cent probability. The results for each data set are shown in Table 2.

**********
Table 2 about here
**********

Table 2 shows the distribution of the error rate of the random classifiers. As can be seen, randomness can explain up to 46 percent, approximately, of out-of-sample errors. Thus, classifiers achieving higher out-of-sample error rates can be considered as random. In fact, only in the first two data sets was Adaboost able to obtain lower out-of-sample error rates. But what are the factors that affect Adaboost's ability to discriminate between stock price movements?

One possible way to answer this question is to gauge traditional benchmarks. In doing so, we can evaluate whether or not simple linear models are able to explain Adaboost's predictability. To that end, we estimated a simple first-order autoregressive model for each period, and the results are shown in Table 3.

**********
Table 3 about here
**********

Table 3 displays the same accuracy measures as Table 1. In addition, Table 3 shows the AR(1) model estimated in the training sample of each data set. Not surprisingly, a simple autoregressive model is able to obtain very similar direction-of-change predictability as Adaboost. Similar to Table 1, the autoregressive models are able to obtain in-sample predictability but fail to detect out-of-sample predictability in the last two data sets. The disappearance of the predictability documented here is consistent with Allen and Karjalainen [23] 's results.

## 4. – Concluding remarks

In this paper, we have implemented a classifier induction approach to analyze the sample evidence on return predictability. We obtain the following general results. First, periods characterized by high first-order serial correlation in stock returns allow both in-sample and out-of-sample direction-of-change predictability. Specifically, a powerful machine learning algorithm called *Adaboost* is able to find a stable function which discriminates, better than randomly made decisions, between upward and downward movements.

Second, Adaboost does over-fit. Functions induced in periods characterized by the lack of autocorrelation in stock returns are able to obtain in-sample predictability but fail to detect out-of-sample predictability. In fact, in many cases, Adaboost's out-of-sample performance decreases as more iterations are run. We have also examined different Adaboost specifications, such as using 4- and 8-node tree-based models instead of stumps, and achieved faster over-fitting.

There are several natural extensions to our analysis. First, machine learning algorithms can be used to examine large price change predictability. They can also be modified to study predictability of large absolute price movements, which are useful for option trading strategies. Second, machine learning algorithms are sufficiently flexible to examine the performance of nested models. For example, one can induce classifiers for small-cap indices using small-cap's or large-cap's lags, and evaluate the lead-lag effect in terms of movement predictability. Finally, machine learning algorithms can be used to identify risk exposures. For instance, we can codify costly lower-tail outcomes and search for "inputs" or "explanatory" variables that help a machine learning algorithm discriminate between the costly lower-tail outcomes and the remainder of outcomes. We hope to explore these issues more fully in future research.

## **Appendix:** The Bias-variance decomposition of 0/1 loss function

The 0/1 loss function is usually the main criterion for classification problems, and may be represented as in the following equation:

$$E_i = L(t_i, \hat{y}_i),$$

where at time *i*, $t_i$ is the "output" or "response" variable $t \in C$, where $C$ is a set of class labels. In this paper, $C \in \{0,1\}$, where $C$ equals to 1 if the observed equity premium is higher than zero, 0 otherwise. $\hat{y}_i$ is the predicted movement. $\hat{y}_i$ equals to 1 if the predicted equity premium is higher than zero, 0 otherwise. $E_i$ equals to 1 if $t_i \neq \hat{y}_i$, 0 otherwise. In other words, the 0/1 loss function represents one less the proportion of correctly predicted signs.

In the machine learning literature, the bias-variance decomposition is widely used as key tool for understating function approximation algorithms. Following Domingos [24] and Valentini and Dietterich [25], bias and variance in a noise-free setting can be defined in terms of the main prediction. The main prediction $y_m$ can be defined as the movement that is predicted more often in the test sample. Thus, the bias (systematic loss incurred by the function) at time *i* can be computed as,

$$B_i = \begin{cases} 1 \text{ if } y_m \neq t_i \\ 0 \text{ if } y_m = t_i \end{cases}.$$

To distinguish between the two different effects of the variance on the loss 0/1 loss function, Domingos [24] defines the unbiased variance, $V_u$, to be the variance when $B_i = 0$, and can be calculated as,

$$V_u^i = \|(y_m = t_i) \text{ and } (y_m \neq \hat{y}_i)\|,$$

where $\|s\| = 1$ if s is true, 0 otherwise. The unbiased variance evaluates the extent to which the estimated function deviates from the correct predictions. The biased variance, $V_b^i$, occurs when $B_i = 1$, and evaluates the extent to which the estimated function deviates from the incorrect predictions. The biased variance can be estimated as,

$$V_b^i(\mathbf{x}) = \|(y_m \neq t_i) \text{ and } (y_m \neq \hat{y}_i)\|.$$

To obtain the loss associated with a given observation a time *i* [denoted by $E_i$], we simply compute the algebraic sum of bias, unbiased and biased variance as,

$$E_i = B_i + V_u^i - V_b^i.$$

In order to compute the aforementioned variables in a test set, we simply obtain the average for each variable. Clearly, if we want a good function that distinguishes between up-and-down movements, we want the bias and the unbiased variance to be small.

## References


[1] E. Fama, The behavior of stock market prices, Journal of Business 30 (1965), 34-105.

[2] A. W. Lo, A. C MacKinlay, Stocks market prices do not follow random walks: Evidence from a simple specification test, Review of Financial Studies 1 (1988) 41-66.

[3] J. Conrad, G. Kaul, Time varying expected returns, Journal of Business 61 (1988) 409-425.

[4] N. Jegadeesh, Evidence of predictable behavior of security returns, Journal of Finance 45 (1990) 881-898.

[5] G. Kaul, Predictable components in stock returns, in: G. S. Maddala, C. R. Rao, eds, Handbook of Statistics, Vol. 14, (Elsevier Science B.V., Amsterdam, 1996).

[6] C. Apte, S. Hong, Predicting equities returns from securities data with minimal rule generation, in: U. M. Fayyad, G. Piatetsky-Shapiro, P. Smyth, R. Uthurusamy, eds., Advances in Knowledge Discovery and Data Mining (AAAI Press/The MIT Press, Cambrigde, MA, 1995).

[7] R. Tsaih, Y. Hsu, C. C. Lai, Forecasting S&P 500 stock index futures with a hybrid AI system, Decision Support System 23 (1998) 161-174.

[8] S. Zemke, Nonlinear index prediction, Physica A 269 (1999) 177-183.

[9] A. Chen, M. T. Leung, H. Daouk, Application of neural networks to an emerging financial market: Forecasting and trading the Taiwan stock index, Computers and Operations Research 30 (2003), 901-923.

[10] K. Kim, Financial time series forecasting using support vector machines, Neurocomputing 55 (2003) 307-319.

[11] P. N. Rodriguez, A. Rodriguez, Predicting stock market indices movements, in: M. Costantino, C. Brebbia, eds., Computational Finance and its Applications (Wessex Institute of Technology, Southampton, 2004).

[12] N. O'Connor, M. G. Madden, A neural network approach to predicting stock exchange movements using external factors, Knowledge-Based Systems 19 (2006) 371-378.

[13] P. Bossaerts, P. Hillion, Implementing statistical criteria to select return forecasting models: What do we learn?, Review of Financial Studies 12 (1999), 405-28.

[14] R. E. Schapire, The strength of weak learnability, Machine Learning 5 (1990) 197-227.

[15] Y. Freund, Boosting a weak learning algorithm by majority, Information and Computation 121 (1995) 256-285.

[16] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, Journal of Computer and System Sciences 55 (1997) 119-139.

[17] T. Hastie, R Tibshirani, J. H. Friedman, The Elements of Statistical Learning: Data Mining, Inference and Prediction (Springer-Verlag, New York; 2001).

[18] M. Culp, K. Johnson, G. Michailidis, ada: an R package for boosting, Journal of Statistical Software 17 (2006), issue 2.

[19] L. Breiman, J. H. Friedman, R. A. Olshen, C. J. Stone, Classification and Regression Trees (Wadsworth, Belmont, CA, 1984).

[20] J. R. Quinlan, Induction of decision trees, Machine Learning 1 (1986) 81-106.

[21] J. R. Quinlan, C4.5: Programs for Machine Learning (Morgan Kaufmann: San Mateo, CA; 1993).

[22] S. R. Safavian, D. Landgrebe. A survey of decision tree classifier methodology, IEEE Transactions on Systems, Man, and Cybernetics 21 (1991) 660-674.

[23] F. Allen, R Karjalainen, Using genetic algorithms to find technical trading rules, Journal of Financial Economics 51(1999) 245-271.

[24] P. Domingos, A unified bias-variance decomposition for zero-one and squared loss, in: Proceedings of the Seventeenth National Conference on Artificial Intelligence (AAAI Press: Austin, TX, 2000) 564-569.

[25] G. Valentini, T. G. Dietterich, Bias-variance analysis of support vector machines for the development of SVM-based ensemble methods, Journal of Machine Learning Research 5(2004) 725-775.

**Table 1**. Direction-of-change predictability of Adaboost.

**A**. First sub-sample. Training (19620807-19681112) and Testing (19681113-19721229)

| Specification | In-sample evidence | | | | Out-of-sample evidence | | | |
|---|---|---|---|---|---|---|---|---|
| | Error | Bias | Vu | Vb | Error | Bias | Vu | Vb |
| M =1 | 0.4136 | 0.4405 | 0.2427 | 0.2697 | 0.3895 | 0.4715 | 0.2246 | 0.3066 |
| M = 2 | 0.4136 | 0.4405 | 0.2427 | 0.2697 | 0.3895 | 0.4715 | 0.2246 | 0.3066 |
| M = 25 | 0.3911 | 0.4405 | 0.177 | 0.2260 | 0.4136 | 0.4715 | 0.1793 | 0.2372 |
| M = 50 | 0.3898 | 0.4405 | 0.175 | 0.2261 | 0.4127 | 0.4715 | 0.1765 | 0.2353 |
| M = 200 | 0.3866 | 0.4405 | 0.117 | 0.1708 | 0.4147 | 0.4715 | 0.1369 | 0.1938 |
| M = 1000 | 0.3230 | 0.4405 | 0.112 | 0.2293 | 0.4272 | 0.4715 | 0.1745 | 0.2189 |

**B**. Second sub-sample. Training (19730102-19790808) and Testing (19780809-198312

| Specification | In-sample evidence | | | | Out-of-sample evidence | | | |
|---|---|---|---|---|---|---|---|---|
| | Error | Bias | Vu | Vb | Error | Bias | Vu | Vb |
| M =1 | 0.4353 | 0.4994 | 0.0803 | 0.1445 | 0.4658 | 0.4766 | 0.1061 | 0.1169 |
| M = 2 | 0.4353 | 0.4994 | 0.0803 | 0.1445 | 0.4658 | 0.4766 | 0.1061 | 0.1169 |
| M = 25 | 0.4107 | 0.4994 | 0.21 | 0.2992 | 0.4604 | 0.4766 | 0.2437 | 0.2599 |
| M = 50 | 0.4053 | 0.4994 | 0.174 | 0.2686 | 0.4667 | 0.4766 | 0.2221 | 0.2320 |
| M = 200 | 0.3783 | 0.4994 | 0.181 | 0.3022 | 0.4739 | 0.4766 | 0.2338 | 0.2365 |
| M = 1000 | 0.3237 | 0.4994 | 0.153 | 0.3291 | 0.4937 | 0.4766 | 0.2572 | 0.2401 |

**C**. Third sub-sample. Training (19840103-19900808) and Testing (19900809-19941230

| Specification | In-sample evidence | | | | Out-of-sample evidence | | | |
|---|---|---|---|---|---|---|---|---|
| | Error | Bias | Vu | Vb | Error | Bias | Vu | Vb |
| M =1 | 0.5494 | 0.4506 | 0.5494 | 0.4506 | 0.5198 | 0.4802 | 0.5198 | 0.4802 |
| M = 2 | 0.5494 | 0.4506 | 0.5494 | 0.4506 | 0.5198 | 0.4802 | 0.5198 | 0.4802 |
| M = 25 | 0.4446 | 0.4506 | 0.029 | 0.0348 | 0.4847 | 0.4802 | 0.0207 | 0.0162 |
| M = 50 | 0.4230 | 0.4506 | 0.081 | 0.1084 | 0.4838 | 0.4802 | 0.0791 | 0.0755 |
| M = 200 | 0.3913 | 0.4506 | 0.116 | 0.1756 | 0.4784 | 0.4802 | 0.1214 | 0.1232 |
| M = 1000 | 0.3355 | 0.4506 | 0.104 | 0.2187 | 0.4703 | 0.4802 | 0.1466 | 0.1565 |

**D**. Fourth sub-sample. Training (19950103-20001226) and Testing (20011227-2004123

| Specification | In-sample evidence | | | | Out-of-sample evidence | | | |
|---|---|---|---|---|---|---|---|---|
| | Error | Bias | Vu | Vb | Error | Bias | Vu | Vb |
| M =1 | 0.5410 | 0.4590 | 0.5410 | 0.4590 | 0.5084 | 0.4916 | 0.5084 | 0.4916 |
| M = 2 | 0.5410 | 0.4590 | 0.5410 | 0.4590 | 0.5084 | 0.4916 | 0.5084 | 0.4916 |
| M = 25 | 0.4187 | 0.4590 | 0.13 | 0.1700 | 0.4945 | 0.4916 | 0.1440 | 0.1410 |
| M = 50 | 0.3981 | 0.4590 | 0.083 | 0.1442 | 0.5055 | 0.4916 | 0.1470 | 0.1331 |
| M = 200 | 0.3724 | 0.4590 | 0.101 | 0.1872 | 0.4985 | 0.4916 | 0.1698 | 0.1629 |
| M = 1000 | 0.3247 | 0.4590 | 0.1210 | 0.2553 | 0.4826 | 0.4916 | 0.1917 | 0.2006 |

**Table 2**. Random predictability

**A**. First sub-sample. Training (19620807-19681112) and Testing (19681113-19721229)

| Percentil | 0.01 | 0.05 | 0.25 | 0.50 | 0.75 | 0.95 | 0.99 |
|---|---|---|---|---|---|---|---|
| In-sample error | 0.4701 | 0.4804 | 0.4907 | 0.4997 | 0.5080 | 0.5202 | 0.5292 |
| Out-of-sample error | 0.4648 | 0.4754 | 0.4899 | 0.4995 | 0.5092 | 0.5236 | 0.5362 |

**B**. Second sub-sample. Training (19730102-19790808) and Testing (19780809-19831230

| Percentil | 0.01 | 0.05 | 0.25 | 0.50 | 0.75 | 0.95 | 0.99 |
|---|---|---|---|---|---|---|---|
| In-sample error | 0.4724 | 0.4802 | 0.4922 | 0.5000 | 0.5078 | 0.5204 | 0.5288 |
| Out-of-sample error | 0.4685 | 0.4757 | 0.4901 | 0.5009 | 0.5108 | 0.5234 | 0.5324 |

**C**. Third sub-sample. Training (19840103-19900808) and Testing (19900809-19941230)

| Percentil | 0.01 | 0.05 | 0.25 | 0.50 | 0.75 | 0.95 | 0.99 |
|---|---|---|---|---|---|---|---|
| In-sample error | 0.4727 | 0.4805 | 0.4924 | 0.4997 | 0.5075 | 0.5195 | 0.5261 |
| Out-of-sample error | 0.4649 | 0.4748 | 0.4901 | 0.5009 | 0.5108 | 0.5252 | 0.5342 |

**D**. Fourth sub-sample. Training (19950103-20001226) and Testing (20011227-20041231)

| Percentil | 0.01 | 0.05 | 0.25 | 0.50 | 0.75 | 0.95 | 0.99 |
|---|---|---|---|---|---|---|---|
| In-sample error | 0.4702 | 0.4795 | 0.4914 | 0.4993 | 0.5099 | 0.5212 | 0.5284 |
| Out-of-sample error | 0.4667 | 0.4747 | 0.4886 | 0.4995 | 0.5104 | 0.5263 | 0.5353 |

**Table 3.** Direction-of-change predictability of an AR(1) model.

**A**. First sub-sample. Training (19620807-19681112) and Testing (19681113-19721229)
*Model:* $R_t = 0.0005313117 + 0.1417891R_{t-1}$

| In-sample evidence | | | | Out-of-sample evidence | | | |
|---|---|---|---|---|---|---|---|
| Error | Bias | Vu | Vb | Error | Bias | Vu | Vb |
| 0.4248 | 0.4402 | 0.0790 | 0.094 | 0.4011 | 0.4715 | 0.0848 | 0.1552 |

**B**. Second sub-sample. Training (19730102-19790808) and Testing (19780809-19831230)
*Model:* $R_t = 0.0001423972 + 0.2224826R_{t-1}$

| In-sample evidence | | | | Out-of-sample evidence | | | |
|---|---|---|---|---|---|---|---|
| Error | Bias | Vu | Vb | Error | Bias | Vu | Vb |
| 0.4259 | 0.4997 | 0.1968 | 0.271 | 0.4676 | 0.4766 | 0.2185 | 0.2275 |

**C**. Third sub-sample. Training (19840103-19900808) and Testing (19900809-19941230)
*Model:* $R_t = 0.0006423667 + 0.05325351R_{t-1}$

| In-sample evidence | | | | Out-of-sample evidence | | | |
|---|---|---|---|---|---|---|---|
| Error | Bias | Vu | Vb | Error | Bias | Vu | Vb |
| 0.4514 | 0.4502 | 0.0330 | 0.032 | 0.4838 | 0.4802 | 0.0234 | 0.0198 |

**D**. Fourth sub-sample. Training (19950103-20001226) and Testing (20011227-20041231)
*Model:* $R_t = 0.003193694 + 0.0008291922R_{t-1}$

| In-sample evidence | | | | Out-of-sample evidence | | | |
|---|---|---|---|---|---|---|---|
| Error | Bias | Vu | Vb | Error | Bias | Vu | Vb |
| 0.4586 | 0.4586 | 0.0000 | 0.0000 | 0.4916 | 0.4916 | 0.0000 | 0.0000 |