Predicting the Stock Market

Thomas Hellström[1] and Kenneth Holmström[2]
Center of Mathematical Modeling (**CMM**)
Department of Mathematics and Physics
Mälardalen University
P.O.Box 883
S-721 23 Västeras, Sweden

Published as Opuscula ISRN HEV-BIB-OP-26-SE[3]

August 12, 1998

Abstract

This paper presents a tuturial introduction to predictions of stock time series. The various approaches of technical and fundamental analysis is presented and the prediction problem is formulated as a special case of inductive learning. The problems with performance evaluation of near-random-walk processes are illustrated with examples together with guidelines for avoiding the risk of data-snooping. The connections to concepts like "the bias/variance dilemma", overtraining and model complexity are further covered. Existing benchmarks and testing metrics are surveyed and some new measures are introduced.

Keywords: Finance, Stock returns, Prediction,Machine learning, Data Mining, Bias/variance, Overtraining, Data-snooping.

[1]also Umeå University Umeå Sweden. email: thomash@cs.umu.se
[2]hkh@mdh.se
[3]Also available as Technical Report Series IMa-TOM-1997-07

Contents

1 What's So Special About Predictions of Stocks?

Stock time series have a number of specific properties that, although not unique when examined one by one, together make the prediction task rather unusual and calls for special considerations when developing and evaluating a prediction system.

- Prediction of stocks is generally believed to be a very difficult task. The most common viewpoint, especially among academics, is that the task of predicting stocks is comparable to that of inventing a *perpetuum mobile* or solving problems like the *quadrature of the circle.*
- The process behaves very much like a random-walk process. The autocorrelation for day to day changes is very low. Hawawini and Keim [18] conclude on several studies that the serial correlation in stock price time series is economically and statistically insignificant.
- The process is "regime shifting", in the sense that the underlying process is time varying. The noise level and volatility in the time series change as the stock markets move in and out of periods of "turbulence", "hausse" and "baisse". This causes great problems for traditional algorithms for time series predictions.

The obvious complexity of the problem and the strong resemblance to random-walk processes has, as we shall see in the following chapters, direct implications on the prediction task, the evaluation task and the application task.

On the positive side, the *application* of a prediction system for stock prices is somewhat special:

- A successful prediction algorithm does not have to provide predictions for all points in the time series. The important measure of success is the hit rate and the generated profit at the points where the algorithm produces predictions. The behavior in between these points is not equally interesting. A missed opportunity to profit doesn't mean lost money!

  The near random-walk behavior also implies a somewhat different *evaluation* situation than normally encountered:
- The evaluation of a prediction algorithm must be done with great care to avoid both 'Type I errors" (failing to accept a correct hypothesis) and "Type II errors" (accepting a false hypothesis). The problem is discussed in more detail in Section 6.

Another interesting property of this problem area is that the prediction algorithms themselves get integrated in the data generating process. Assume as an example that a new sophisticated algorithm shows a superior and unquestionable predictive power when applied to both historical data and real trading. As this technique becomes public knowledge and commonly used, the market participants will include the new algorithm as a new tool together with all previously suggested and used stock evaluation methods. The predictive power of the new algorithm will then decrease until it becomes useless. Viewed this way, what remains to make profit on in trading is no more than the residual from the superimposed prediction algorithms that comprise the process of price generation. If the superimposed algorithms do a good job, the residual should be indeed the near-random walk that we are facing.

1.1 Common Viewpoints

It is very easy to get opinions from people about what "has to be" successful strategies in stock trading. It is equally difficult, by reasoning, to show that a proposed strategy is not successful even if that really is the case. As we shall see in Section 6, it is very likely that a random strategy appears to be successful even if tested for a seemingly long period.

1.1.1 The Efficient Market Hypothesis

The efficient market hypothesis (EMH) states that the current market price reflects the assimilation of all the information available. This means that given the information, no prediction of future changes in the price can be made. As new information enters the system the unbalanced state is *immediately* discovered and quickly eliminated by a "correct" change in market price. Depending on the type of information considered, there exist three forms of EMH:

The weak form: Only past price data is considered. This kind of EMH rules out any form of predictions based on the price data only, since the prices follow a *random walk* in which successive changes have zero correlation[1].

- The semistrong form: All publicly available information is considered. This includes additional trading information such as volume data (e.g. number of traded stocks) and fundamental data such as profit prognoses and sales forecasts.
- The strong form: All information, publicly as well as privately available, is considered.

The application of the weak and semistrong forms of EMH has been fairly well supported in a number of research studies, e.g. White [44], Lowe and Welsh [26], and has for long been the "official" viewpoint in the academic society. However, in recent years many published reports show that the efficient market hypothesis is far from correct. Fama goes as far as stating, in a review, that the efficient market hypothesis surely must be false [13].

The strong form has for obvious reasons been difficult to test statistically due to shortage in available data. However, indirect methods analyzing the performance of mutual fund managers did not prove them to be superior to other investor types [36].

1.1.2 Traders Viewpoint

Despite the traditionally massive support for EMH we have a majority of the market actors believing, or at least claiming, that they can predict the prices in a way that can make

[1]A more strict description of the relation between random walk and the correlation can be found in Section 6.

them profit. Almost all professional traders rely to some extent on either technical or fundamental analysis of past information. They buy when the market is "bullish" and sell when it is "bearish"; thereby assuming a direct correlation between the current trend and future prices.

There are also numerous research papers claiming that applying nonlinear models such as neural networks and genetic algorithms *can* provide successful predictions. However, this may not contradict the EMH if new useful methods are not *immediately* assimilated by the global trading community. LeBaron [23], and Sweeney and Surarjaras [34], have investigated technical trading strategies and found some evidence of a drop off in profit in recent years.

Another reasonable argument against the EMH deals with the different time perspectives different traders have when they do business. For example, a majority stock owner will react quite differently than a floor day trader when a stock suddenly drops in value. These differences in time perspectives will cause anomalies in the market prices even if no new information has entered the scene. It may be possible to identify these situations and actually predict future changes.

In summary, most arguments in favor of the EMH relies on statistical tests showing no predictive power in the tested models and technical indicators. Most arguments against the EMH refer to a time delay between the point when new information enters the system and the point when the information has been assimilated globally and a new equilibrium with a new market price has been reached. Viewed this way, the controversy is only a matter of how the word "*immediately*" in the EMH definition should be interpreted. It may just be the case that traders in general simply are faster than academics.

2 Problem Formulation

2.1 Available Data

A prediction algorithm uses a set of known entities to produce a prediction of future values of the same or other entities. In the case of stock predictions, the entities can be divided into two categories; pure *technical* data and *fundamental* data. The technical data is the only data used by the technical analysts. Their prognoses are based on past stock data only. The fundamental analysts in addition to the pure technical data, include data related to the companies' actual activity and the market situation. In addition to these two categories of data, *derived entities* can be produced by transforming and combining technical and fundamental data. This section describes the most common entities used either as input or output in the prediction of stocks.

2.1.1 Technical Data

The daily available data for each stock is represented in the following four time series with data for each day of trading (normally Monday-Friday):

- y_C or y : close value; price of the last performed trade during the day

- y_H : highest traded price during the day
- y_L : lowest traded price during the day
- V : Volume; total number of traded stocks during the day

Data for each individual trade during the day is sometimes available as well. However, this data is seldom used in data modeling of stock prices, but it is often used for timing purposes before ”pulling the trigger” in real trading.

The most obvious choice of entity to predict is the time series y_C or y. Some of the drawbacks of this approach are:

- The prices y normally vary greatly and make it difficult to create a valid model for a longer period of time.
- y for different stocks may easily differ over several decades and therefor can not be used as the same type of input in a model.

The *returns*, defined in (1) in Section 2.1.3, are therefor often used instead.

2.1.2 Fundamental Data

Apart from the daily sampled data described above, there is a lot of information concerning the activities and financial situation of each company. Most companies that are quoted at a stock market are analyzed on a regular basis by the professional market analysts at the financial institutes. The analyses are often presented as numerical items, which are supposed to give hints on the ”true” value of the company's stock. The buy and sell recommendations are then formed, either intuitively or with some sort of mathematical weighting of these numerical items. A fundamental analysis of a company typically focuses on the following three factors (Achelis [2]):

1 The general economy:

  - inflation.
  - interest rates.
  - trade balance etc.

2 The condition of the industry; The state of the industry to which the company belongs:

  - Other stock prices normally presented as indexes (i.e. weighted means) such as the US ”Dow Jones”, the German ”DAX” and the Swedish ”Generalindex”.
  - The prices of related commodities such as oil, different metals and currencies.
  - The value of the competitors' stocks.

3 The condition of the company. This is extracted from the company's financial statements. From these a number of useful variables can be calculated:

- p/e; The stock price divided by the earning per share during the last 12 months.
- Book value per share; net assets (assets minus liabilities) divided by the total number of shares.
- Net profit margin; net income divided by total sales.
- Debt ratio; liabilities divided by total assets.
- Prognoses of future profits.
- Prognoses of future sales.

Of course, most of the work performed by the professional financial analysts is not publicly available. However, there is plenty of high quality data published weekly in several financial magazines. However, the availability of huge amounts of data in machine readable form is much better for technical data than for fundamental data.

2.1.3 Derived Data

Returns. The **one-step** returns $R(t)$ is defined as the relative increase in price since the previous point in the time series:

$$R(t) = \frac{y(t) - y(t-1)}{y(t-1)}. \tag{1}$$

A common variant is the log-return

$$R(t) = \log \frac{y(t)}{y(t-1)}. \tag{2}$$

The log-returns are often used in academic research while the former version is most common in the trading community. If the natural logarithm is used, the two measures are very similar for small changes, since $\ln(\frac{a}{b}) \sim \frac{a}{b} - 1 = \frac{a-b}{b}$.

One often chooses to predict $R(t)$ instead of the original prices y. Some of the reasons for this are:

1. $R(t)$ has a relatively constant range even if data for many years are used as input. The prices y obviously vary much more and make it difficult to create a valid model for a longer period of time.

2. $R(t)$ for different stocks may also be compared on an equal basis (however, this is seldom done in published research papers).

3. It is easy to evaluate a prediction algorithm for $R(t)$ by computing the prediction accuracy of the sign of $R(t)$. An accuracy above 50% (or more precisely above the mean) indicates that a true prediction has taken place.

   $R(t)$ can be generalized to cover more than one day intervals. The k-step-return $R_k(t)$ is consequently defined as:

$$R_k(t) = \frac{y(t) - y(t-k)}{y(t-k)}. \tag{3}$$

The k-step-return $R_k(t)$ can be interpreted as the k-day price trend for the stock.

Volatility. Volatility is a common concept in financial analysis. It describes the variability in a stock price y and is used as an estimate of investment risk, but also for profit possibilities. There exist several definitions, see the thesis by Andersson [4]. The standard definition is:

$$V = \sqrt{\frac{1}{N-1}\sum_{t=1}^{N}(\ln(\frac{y_t}{y_{t-1}}) - m)} \tag{4}$$

where m is the mean value,

$$m = \frac{1}{N}\sum_{t=1}^{N}\ln(\frac{y_t}{y_{t-1}}). \tag{5}$$

In this case the volatility equals the standard deviation of the log-return series $R(t)$. If the volatility is to be used as input in a model identification, a sliding window is normally used for the sums in the definition. It has been shown empirically that volatility and predictability of financial time series are connected, although a theoretical explanation does not exist [33] .The volatility is of great interest for financial analysts and provides useful information when estimating investment risk in real trading. The volatility may be used both as input and output in prediction models.

Volume. The increase of traded volume V is often viewed as an indication of new information reaching the market. Therefor it may be useful to include the rate of change of V as an additional input variable.

Turning Points. Turning points in a stock price chart can be viewed as positions where equilibrium between demand and supply has been reached. Therefor, they may be viewed as more significant than the data in between, which could be regarded as noise in this context. Therefor, one approach may be to use as input the distance in time and price to the latest turning points.

Technical Indicators. Technical stock analysis deals extensively with derived entities. For a thorough description of the most common technical indicators, see [22]. Some examples are:

- Relative Strength Index (RSI). The relation between the average upward and downward price change within a time window of fixed length. The window is normally 14 days backwards.
- Moving average (MA). A price rise above a moving average is interpreted as a buy signal, and a fall below it is interpreted as a sell signal.

- Moving average convergence divergence (MACD). Two moving averages with different backward windows are used. The difference between the moving averages are also smoothed. Buy and sell signals are generated from the crossings and trends of these filtered price signals.
- Relative strength (RS). The concept of relative strength trading involves selecting stocks for trading by their performance compared to the general market, represented by some relevant index.

Even if the relevance as predictors of most of the standard technical indicators is highly dubious, they may be useful as input in various prediction models.

Artificial Data. Generating artificial data by adding noise to y (or by shifting data slightly) introduces a degree of time invariance. It's a way of telling the inference system to concentrate on the fundamental properties of the data instead of trying to model what really is noise. Bishop [6] has shown that adding noise to data is equivalent to Tikhonov regularization. More information about regularization is found in Section 5.7.

Relative Stock Performance. Instead of attempting to predict individual stock prices, one may address the problem of performing and comparing predictions for many stocks at the same time. The central problem in portfolio management concerns selecting the weights of stocks in a portfolio in such a way that the investment risk for a given level of expected return is minimized. The traditional portfolio theory uses quadratic programming methods to minimize the variance of the returns in the portfolio. New approaches show that investors perceive risk only on the downside part of the return distribution. New concepts such as *target semivariance* and *downside risk* are often used [1].

Another multi-stock approach is to predict the relative levels or a ranking list of a given set of stocks. The rank of a stock is computed as the stock's position in a sorted list of stock returns.

2.1.4 Data Transformations

The input data described above is often transformed before the actual modeling takes place. The purposes of the transformations can be divided into the following categories:

Dimensionality Reduction. In principle, future values in the time series are dependent on all previous values. Therefor, it is sometimes useful to compute aggregated entities to be used as the actual inputs in the modeling. Examples are running means of y, y_H, y_L and V.

Normalization. The purpose of normalization is to reduce range differences in the input variables. Two major types can be distinguished:

1. Reduction of size differences between different input variables to enforce similar behavior upon the modeling algorithms. For example, before entering variables into an

artificial neural network, all variables are often scaled to cover the range 0-1. This scaling, and the corresponding re-scaling of the network output, is often performed automatically in many neural network software packages.

2. Reduction of size differences over time in an input time series. It reduces the non-stationary effect. For example, normalized y, y_H and y_L: The measured time series y, y_H and y_L often varies by several hundred percent if measured for some years. Therefor, it is often necessary to normalize the data to reduce the non-stationary effect. There exist several procedures. Some examples are:
   - Transformation to relative changes (returns) as defined in (1).
   - Transformation to log-returns as defined in (2).
   - Normalization using the mean.
   - Normalization using both the mean and the standard deviation.

As explained in Section 2.1.3, the first two measures are very similar for small daily changes. The motivation for using the log-return is statistical; the variance becomes stabilized and outliers become less dominant. Both transformations are very common in time series predictions of financial data.

The normalization using the mean is defined as

$$y_n(t) = y(t)/y_{mean}(t) \tag{6}$$

where the mean value $y_{mean}(t)$ is computed using a running window of fixed length n, e.g. 30 days backwards. The normalization with the mean and standard deviation is defined as

$$y_n(t) = (y(t) - y_{mean}(t))/\sigma_y(t) \tag{7}$$

where the estimated standard deviation $\sigma_y(t)$, like $y_{mean}(t)$, is computed in a running window of length n. The variable $y_n(t)$ is expressing how many standard deviations the y values may differ from their running mean. Similarly, compute the normalized volume $V_n(t)$ as

$$V_n(t) = (V(t) - V_{mean}(t))/\sigma_V(t) \tag{8}$$

where the mean $V_{mean}(t)$ and the standard deviation $\sigma_V(t)$ of the volume are computed in a running window of fixed length n_V. A normal backward window size is $n_V = 30$ days. $V_n(t)$ is expressing how many standard deviations the volume figure differs from its running mean.

Linearization. It is often advantageous to remove obvious nonlinear factors that would otherwise have to be taken care of in the modeling. One example is the volume $V(t)$ which is often distorted by extremely high figures resulting from individual trades by large institutional traders such as mutual funds. The linearization in this case may be a pre-processing squashing function that suppresses values outside a set boundary. For example, the input $V(t)$ may be transformed to $V'(t)$ according to:

$$V'(t) = \frac{1}{1 + e^{-V(t)}} \tag{9}$$

Principal Component Analysis (PCA). Principal Component Analysis provides an efficient technique for dimensionality reduction of a data set. It works by finding a linear transformation T which is applied to the original data set of p dimensional vectors. The transformed vectors are still p dimensional, but T is computed in such a way that the variances along each of the new p dimensions are sorted in a descending order by their sizes. Therefor, a dimensionality reduction of the data set can be performed by simply ignoring the dimensions with the smallest variances. The mean-squared error in such a truncation can be shown to be the sum of the variances of the truncated dimensions. T is computed by solving an *eigenvalue* problem, commonly encountered in linear algebra. The *eigenvectors* of the correlation matrix R correspond to the axes in the new transformed coordinate system and the *eigenvalues* correspond to the variances measured along each one of these axes. Principal component analysis has been successfully applied for dimensionality reduction in many areas such as image recognition and speech recognition [19].

2.2 Defining the Prediction Task

In this section we will discuss various ways in which the prediction problem can be formulated. The discussion will be restricted to predictions of stock price. It should be noticed that the stock's price is not the only entity of interest for a stock analyst. We will formulate the general prediction task as a special case of Inductive Learning which is defined as follows:

Given a set of N examples,$\{(x_i, z_i), i = 1, N\}$ where $f(x_i) = z_i, \forall i$,return a function g that approximates f in the sense that the norm of the error vector $E = (e_1, ..., e_N)$ is minimized. Each e_i is defined as $e_i = e(g(x), z_i)$ where e is an arbitrary error function.

A general prediction task may be formulated as a special case:

2.2.1 Prediction as Inductive Learning

In the case of prediction problems we define a time order and pick examples from time series: Given a set of N examples, $\{(\mathbf{X}(t), z(t + h)), t = 1, ..., N\}$, where $f(\mathbf{X}(t)) = z(t + h)$, $\forall t$, return a function $g(t)$ that approximates $f(t)$ in the sense that the norm of the error vector $E = (e_1, ..., e_N)$ is minimized. Each e_t is defined as $e_t = e(g(\mathbf{X}(t)), z(t + h))$ where e is an arbitrary error function. Thus we are aiming at approximating $z(t+h)$ with $g(\mathbf{X}(t))$ where h is the prediction horizon. The prediction error in time step t is a function e of example output z, and g applied on the example inputs X.

In order to be more concrete we need to specify the following entities:

- The "Input", i.e. the $\mathbf{X}(t)$ vector.
- The "Output", i.e. the $z(t)$ vector.
- The error function $e(g(\mathbf{X}), z)$
- The error measure, based on some vector norm, for computing $||E||$.
- Prior knowledge of the approximating function g.

It is interesting to note that the function g that we eventually learn depends on all these entities. Even the error function, which often is regarded as a mere measure of goodness of fit, is highly involved in the model selection. By specifying the above five entities a number of general prediction approaches can be expressed:

The Standard Time Series Approach. The input part $\mathbf{X}$ and output part z of each example $(\mathbf{X}(t), z(t+h))$ have the following form in this approach:

$$\mathbf{X}(t) = (y(t), y(t-1), ..., y(t-k+1)) \tag{10}$$

$$z(t+h) = y(t+h). \tag{11}$$

The inputs are thus formed as lagged values of the time series $y(t)$, that we want to predict h steps ahead. In the case of stock predictions, the time series $y(t)$ is typically the returns according to (1) or the closing price each day for the stock. The prediction error e in each step is taken as the difference between what g provides and what the examples suggest: $e_t = g(\mathbf{X}(\mathbf{t})) - z(t+h)$. The root mean square error (RMSE) is used as the vector norm:

$$\|E\| = \sqrt{\frac{1}{N-h-k+1} \sum_{t=k}^{N-h} e_t^2} \tag{12}$$

The bias for the unknown function g can be chosen in many ways. Common choices are:

- A linear autoregressive (AR) model:

$$g(t) = \sum_{m=1}^{d} a_m y(t-m). \tag{13}$$

- A general nonlinear model implemented with a feed forward neural network.

The time series formulation is not always ideal for useful predictions of financial time series for the following reasons:

1. The fixed prediction horizon h does not reflect the way in which financial predictions are being used. The ability of a model to predict should not be evaluated at one single fixed point in the future. A big increase in a stock value 14 days in the future is as good as the same increase 15 days in the future!

2. The underlying processes that generate the time series have dramatically different characteristics for different parts in the series. It is therefor not a very good idea to attempt to fit a global model, however ingenious, for the entire time span. On the other hand, using a sliding window and computing models such as the autoregressive-moving average (ARMA) model within the window is "too narrow minded" and uses no knowledge from data outside the fixed window.

3. The RMSE measure (12) treats all predictions, small and large, as equal. In the case of predicting returns, i.e. $y(t) = R(t)$ in (1), this is not always appropriate. Predictions that would never be used for actual trading (i.e. price changes to0 small to be interesting) may cause higher residuals at the actual points of interest, in order to bring the RMSE to a minimum. One solution to avoid this would be a weighted RMSE measure, where small predicted changes were attenuated or completely cancelled from the right hand side sum in 12. However, this would favor a global zero change prediction ($g(\mathbf{X(t)}) = 0$ for all t), which would give a RMSE equal to zero! However, the possible profit made from such a prediction system would not be overwhelming. Alternative modifications of the error function and the vector norm are possible but have not been further examined in this study.

4. A small predicted change in price, followed by a large real change in the same direction, will be penalized by the RMSE measure. This is not in agreement with the way we judge the quality of a stock prediction in actual trading.

The Trading Rule Approach. Instead of predicting stock prices for all points 1 to N, algorithms can be constructed to recognize situations where one should buy and sell stocks respectively. This approach can also deal with some of the disadvantages of the Standard Time Series approach described above.

The task can be defined as a classifying problem with three classes: "Buy", "Sell" and "Do nothing". Since we want classifications for all points $1, ..., N$, we still need to use the time series framework. A trading rule can be described as a time series $T(t)$ defined as

$$T(t) = \left\{ \begin{array}{lcl} \text{Buy} & : & \text{if } g(\mathbf{X}(t)) > 0 \\ \text{Sell} & : & \text{if } g(\mathbf{X}(t)) < 0 \\ \text{Do nothing} & : & \text{otherwise} \end{array} \right\}. \tag{14}$$

The unspecified function g determines the type of the trading rule. The general problem of generating the function g can be formulated within the general framework of Inductive Learning.

The input part $\mathbf{X}$ of each example $(\mathbf{X}(t), z(t+h))$ has the following form:

$$\mathbf{X}(t) = (R_1(t), ..., R_N(t)), \tag{15}$$

where each $R_k(t)$ is an observable feature at time t. In the case of stock predictions it may be for example the k-day returns as defined in (1) or a standard technical indicator such as the RSI or the MACD.

The output part z of each example has the following form:

$$z(t+h) = R(t+h). \tag{16}$$

The prediction error e in each step is somewhat different from the time series approach:

$$e_t = \left\{ \begin{array}{lrrl} 1: & g(\mathbf{X}(t)) > & \alpha & \text{AND} \quad z(t) < 0 \\ 1: & g(\mathbf{X}(t)) < & -\alpha & \text{AND} \quad z(t) > 0 \\ 0: & \text{otherwise} & & \end{array} \right\} \tag{17}$$

The ordinary RMSE measure (12) is used as the vector norm. The learning task is as before to find a function g that minimizes $\|E\|$ in (12).

This formulation covers the so-called "technical indicators", which are very common in real trading. Even if they are seldom generated in this formal way, they can often be reformulated as a trading rule $T(t)$ as described above. One example is a moving average trading rule, that can be built from two moving average functions operating on the stock prices $y(t)$. Buy and sell signals are generated when two moving averages of different orders intersect. It should be emphasized that the relevance of most of the commonly used technical indicators is highly dubious. Our own studies of the correlation between a number of technical indicator signals and future stock price movements show no significant correlation! However, the general trading rule formulation can be of great interest when developing new prediction algorithms. Since the prediction deviation $\|g(\mathbf{X}(t)) - z(t)\|$ for the "Do nothing" situations is excluded from the error measure, it is possible to develop a model that takes into account the varying noise contents in the data. In this way regions with low predictability can be excluded from the model.

3 Methods

In this section we review some of the techniques commonly used in prediction of time series in general, and financial time series in particular.

3.1 Traditional Time Series Predictions

The beginning of traditional time series analysis might have been in the end of the 20's when Yule invented the autoregressive method (AR) in order to predict sunspots. The general AR-model expresses future values of the time series as a linear combination of past values plus a random noise component:

$$y(t) = \sum_{m=1}^{d} a_m y(t-m) + e(t) \tag{18}$$

Another common model for time series analysis is the moving average model (MA):

$$y(t) = \sum_{n=1}^{M} b_n e(t-n) \tag{19}$$

The above equation describes a situation where the time series y is controlled by an external time series e in a linear and deterministic relation.

Combining the two equations yields the autoregressive-moving average (ARMA) model, which dominated the time series analysis for more then 50 years. An important step beyond these global linear models was taken by Tong and Lim in 1980 when they suggested the *threshold autoregressive model* (TAR). TAR consists of two AR models which are selected locally depending on the system's state.

3.2 Machine Learning

Several methods for inductive learning have been developed under the common label "Machine Learning". All inductive methods use a set of examples to generate an approximation of the underlying function that generated the data. Each example comprises an input vector and an associated output vector. The aim is to draw conclusions from the examples in such a way that when new, previously unseen, input vectors are presented to the system, the output vectors can be successfully predicted.

3.2.1 Nearest Neighbor Techniques

The method of k-nearest-neighbors is a general classification technique that makes minimal assumptions about the underlying function to be modeled. To classify a point p one simply finds the set of k closest points from the example set. In the case of time series the input points p_t are typically formed by picking consecutive values from the time series y; $p_t = (y(t-1), y(t-2)..., y(t-d))$. In the general case, the input points p_i can be any type of feature vector that is believed to have predictive power. We have conducted extensive tests with feature vectors consisting of stock returns R_k defined in (3). For example, $p_t = (R_1(t), R_5(t), R_{10}(t), R_{20}(t))$.

In stock prediction, the classification $C_k(t)$ for the points is typically the sign of the k-day return computed k days ahead (i.e. $R_k(t+k)$):

$$C_k(t) = \left\{ \begin{array}{rcl} +1 & : & \text{if} \quad y(t+k) > y(t) \\ -1 & : & \text{if} \quad y(t+k) < y(t) \\ 0 & : & \text{if} \quad y(t+k) = y(t) \end{array} \right\}. \qquad (20)$$

The closeness is normally computed as the Euclidean distance in the input space. The mean or median classification of the k closest points is then taken as an estimate of the classification for point p . In this way we can immediately produce classifications given a set of examples. There exist a multitude of variants of the basic algorithm. A discussion of weighting schemes can be found in Robinson [32]. For an early work on time series applications and proofs of convergence, see e.g. Yakowitz [46].

Even if the k-nearest-neighbors algorithm is computationally expensive in the application phase, it is very attractive in initial data analysis, where questions about predictability and input variable selection are the important issue. It can be argued that failure in applying the k-nearest-neighbors algorithm to a specific problem implies that the problem can not be solved with *any* inductive method. The sole assumption made in the method is that close inputs are mapped to close outputs. It's hard to see how a sufficient amount of data from *any* continuous function should fail in such a test. The conclusion in such a case would be that a functional relation between the selected input and the output can *not* be shown given the available data, without imposing further restrictions on the functional relationship. However, other methods using *stronger* models (described further in Section 5.3), may be more successful than the totally unbiased k-nearest-neighbors algorithm. One must also realize that k-nearest-neighbor in a normal implementation is a *global* method. In the search for nearest neighbors one either scans the entire training set or all previous points in the training set. The latter method is used to avoid peeping into the future when predicting a point classification. In either case, the neighbors are picked from a time period that may very well be too long if the underlying function is non-stationary. Weighting schemes or windowing techniques may be useful in such cases.

3.2.2 Neural Networks

Neural networks are typically used for pattern recognition tasks. A set of examples is presented as input for the network which is trained to assign each example to one or more classes. The examples consist of a number of features and an associated class assignment. Neural networks are often also used for nonlinear regression, where the task is to find a smooth approximation between multi-dimensional points. In both cases, the input and output data is presented simultaneously to the network. In the case of predictions of time series, the situation is somewhat different. No patterns with features are available. Instead, future time series values should be modeled as a function of previous values. This temporal dimension can be incorporated into the network setup in a number of ways:

- The lagged values of $y(t)$, i.e. $y(t), y(t-1), y(t-2), ...$, can be used to construct a feature vector, which is input to the neural network. This can be viewed as a nonlinear generalization of the AR method previously described. The network output can be written as:
$$O(t) = g[y(t-1), y(t-2)..., y(t-d)], \tag{21}$$
where g is a nonlinear function that is created in the training of the network. As a special case $g(y) = \sum_{m=1}^{d} a_m y(t-m) + e(t)$ is recognized as the well-known AR model. The motivations for this nonlinear generalization is the following. First, *Takens Theorem* [35] states that for a wide class of deterministic systems, there exists a *diffeomorphism* (one-to-one mapping) between past values $y(t-1), y(t-2)..., y(t-d)$ of the time series and the state of the system at time t. This further implies that there exists a function g so that $y(t) = g[y(t-1), y(t-2)..., y(t-d)]$. Second, it has been shown that a neural network is an *universal approximator*, capable of approximating any continuous function [8]. These statements together give the basic motivation for the use of artificial neural networks in time series prediction.

- Recurrent neural networks have an architecture where the outputs from intermediate neurons are fed back to the input layer. In this way the network will contain memory of previous values of the input data. A single value $y(t-1)$ is used as input and a single value $O(t)$ is produced as output from the neural network. The temporal dependencies are modeled in the weights in the feedback loops in the network. Applications of recurrent networks can be found in Ellman [10].

- Another sophisticated method replaces the connections between nodes in a feed-forward neural network with FIR filters. A FIR-filter is a moving average filter with input x and output y:

$$y(t) = \sum_{n=1}^{M} b_n x(t-n). \tag{22}$$

In this way memory of old values is stored in each FIR filter. The number of necessary weights can sometimes be significantly reduced by assuming an implicit symmetry in the optimal network structure[38].

Neural Networks as Classifiers. Neural networks are also often used for classification problems where p-dimensional input vectors are to be mapped to a m-dimensional output vector with exactly *one* element set to *one* and the rest set to *zero*. The position of the element set to one represents the class associated with the corresponding input vector. In this way, examples of input vectors and corresponding output vectors can be used to train the network to assign a class label to unknown input vectors. It is typically implemented with a multilayer perceptron with p inputs and m outputs. The output layer often uses the sigmoid as an activation function producing a m-dimensional output vector with values in the closed interval $[0,1]$. It has been shown (see e.g. [19]) that the output from a successfully trained network is an asymptotic approximation of the *a posteriori* class probabilities. This means that the optimum classification of an input vector is the position in the output vector that has the highest value. It is common to offset the target output vectors by a small amount ε such that the zero-elements are represented by ε and the one-element by $1-\varepsilon$. The reason for this is the need to avoid saturation of the activation function in the training process. In this case the output from the trained network can not be directly interpreted as a probability. A linear transformation $[\varepsilon, 1-\varepsilon] \rightarrow [0,1]$ must be first applied. However, the position in the output vector that has the highest value is still the optimum classification for a given input vector.

The "Capacity" of a Neural Network. An interesting remark about the relation between polynomial regression and regression with neural networks is given by Gershenfeld and Weigend in [16]. The complexity of a polynomial is increased by introducing pairs $(x_1x_2, x_1x_3, ...)$ of the input vector $(x_1, x_2, x_3, ...)$. Each new pair has associated weights to be estimated in the regression. Cover (1965) has shown that both a neural network and a polynomial have a "capacity" proportional to the number of parameters, i.e. a neural network can not express "more" functions then a polynomial with the same number of parameters. The difference, that is in favor of the neural networks, is that the order of interactions between inputs has to be chosen explicitly in the case of adding new parameters to a polynomial model. The neural network has the power to find the suitable order of interactions in the learning process. Adding a weight simply increases the *number* of interactions and not the *order* of interactions.

3.3 Technical Stock Analysis

The term *technical analysis* denotes a basic approach to stock investing where the past prices are studied, using charts as the primary tool. The roots of technical analysis go back to the Dow theory, developed in the beginning of this century by Charles Dow[2]. The basic principles include concepts such as the trending nature of prices, confirmation and divergence, support/resistance and the effect of traded volume. Many hundreds of methods for prediction of the stock prices have been developed and are still being developed on the grounds of these basic principles. Most of the developed prediction methods have very weak scientific support and it is probably fair to say that the authors of the numerous books on Technical Stock Analysis have made more money selling their books than either themselves or their readers have made applying the theories in practice. A thorough survey of the most common technical indicators can be found in [2].

4 Evaluating Performance

The correct way to evaluate prediction performance depends on the approach chosen for the problem formulation as described in Section 2.2. We will start by describing some common benchmarks and then some common error measures. Further notes regarding evaluation are discussed in Section 6.3.

4.1 Benchmarks

The need for good benchmarks when evaluating stock prediction algorithms can not be overestimated. What is good and bad performance depends on the alternatives. Even a prediction algorithm that loses money for some period of time may turn out to be successful when compared to other alternatives. We will discuss three benchmarks commonly used for evaluating prediction algorithms.

4.1.1 Naive Prediction of Stock Prices

The naive prediction asserts today's stock price as the best estimate of tomorrows price. This is a direct consequence of the random-walk hypothesis. It is always a good idea to measure the goodness of a predictor in relation to this trivial predictor. This is done in *Theil coefficient of inequality* described below as T_p in Section 4.2.1.

4.1.2 Naive Prediction of Returns

The naive prediction of stock returns asserts today's return (increase since yesterday) as the best estimate of tomorrows return . This naive prediction is formed from the observation of a one-step memory in the price generating process. In [20] we show that the autocorrelation of stock returns exhibits a significant first lag component that indicates a correlation between adjacent returns. It is a good idea to measure the goodness of a predictor in relation to this trivial predictor. This is the purpose of our suggested measure "Relative hit rate", denoted H_0 and described in Section 4.2.2 below.

[2]The widely used Dow Jones Industrial Average followed as a direct result of Charles Dow's new methods of analyzing the markets.

4.1.3 Buy-and-Hold

The *Buy-and-hold* test finds out whether the net profit result using a prediction algorithm or strategy is due to prediction accuracy or merely to a general market trend. The buy-and-hold return r_b for a time period $\{t...t+n\}$ of a stock is defined as

$$r_b = \frac{(y_{t+n} - y_t)}{y_t}, \tag{23}$$

i.e. the profit made when buying at the start and selling at the end of the time period.

4.2 Metrics for Testing

The general metrics for testing work by comparing the predicted values to the actual outcome in the training data and ultimately in the test set. Refenes [30] has presented a survey of a large number of measures of performance. In this section we present those we have found essential as well as some new important metrics for serious evaluation of financial prediction algorithms.

4.2.1 Theil Coefficient

Predicting Stock Prices. The predictions of stock prices for time t are expressed by the time series $\{\hat{y}(t), t = 1, ..., N\}$. The actual prices are denoted by the time series $\{y(t), t = -d+1, -d+2, ..., 0, 1, ..., N\}$. It is assumed that the d first points in $y(t)$ are used in the initial upstart phase.

The Theil coefficient of inequality provides a measure of the model performance relative to the naive price predictor.

$$T_p = \frac{\sqrt{\sum_{t=1}^{N} (y(t) - \hat{y}(t))^2}}{\sqrt{\sum_{t=1}^{N} (y(t) - y(t-1))^2}} \tag{24}$$

For $T_p > 1$ the predictor is worse than the naive price predictor while $T_p < 1$ implies that the predictor is making better predictions. The naive predictor asserts that the best estimate of tomorrow's price $y(t+1)$ is today's price $y(t)$.

The Theil coefficient is often referred to as *the information coefficient* or the *t-test.*

Predicting Returns. The Theil coefficient (24) compares the performance to that of the naive price predictor. We propose a similar measure to compare to the performance of the naive return predictor. The predictions of returns at time t are denoted by the time series $\{\hat{R}(t), t = 1, ..., N\}$. The actual returns are denoted by the time series $\{R(t), t = 1, ..., N\}$.

The Theil coefficient for returns provides a measure of the prediction performance relative to the naive return predictor.

$$T_r = \frac{\sqrt{\sum_{t=1}^{N} (R(t) - \hat{R}(t))^2}}{\sqrt{\sum_{t=1}^{N} (R(t) - R(t-1))^2}} \tag{25}$$

For $T_r > 1$ the predictor is worse than the naive return predictor while $T_r < 1$ implies that the predictor is making better predictions.

4.2.2 Hit Rate

The hit rate of a predictor indicates how often the sign of the return, i.e. the daily change defined in (1), is correctly predicted. It is computed as the ratio between the number of correct non-zero predictions $\hat{R}(t)$ and the total number of non zero moves in the stock time series.

$$H = \frac{\left|\left\{t|R(t)\hat{R}(t) > 0, t = 1, ..., N\right\}\right|}{\left|\left\{t|R(t)\hat{R}(t) \neq 0, t = 1, ..., N\right\}\right|} \tag{26}$$

$\hat{R}(t)$ is the prediction of $R(t)$ computed at time $t-1$. The norm of the set in the definition is simply the number of elements in the set.

In analogy with the Theil coefficient the corresponding measure for the naive return predictor is proposed:

$$H_N = \frac{|\{t|R(t)R(t-1) > 0, t = 1, ..., N\}|}{|\{t|R(t)R(t-1) \neq 0, t = 1, ..., N\}|} \tag{27}$$

Finally, the ratio between the two hit rates provides the predictor's hit rate performance relative to the naive return predictor:

$$H_0 = \frac{H}{H_N} \tag{28}$$

This entity H_0 which we call "Relative hit rate", compares the hit rate of the predictor to that of the naive return predictor. For $H_0 < 1$ the predictor is worse than the naive return predictor, while $H_0 > 1$ implies that the predictor is making better predictions.

The reason for avoiding both zero predictions and zero returns in the computation of the hit rate is the following. If zeros were included, we would have to decide whether the following five combinations should be regarded as "hits" or not: zero prediction/outcome>0, zero prediction/outcome<0, zero prediction/zero outcome, prediction>0/zero outcome, prediction<0/zero outcome. Whatever choice is made for the classification of these situations, a non- symmetric treatment of the positive and negative returns will result. Since the zero returns can account for more than 20% of the samples in typical stock data, see [20], they would result in "Up fractions" arbitrarily either greater than or less than 50%. Such a result would conceal the random-walk nature of the time series. By removing all zeros from both predictions and outcomes, "Up fractions" very close to 50% are achieved.

4.2.3 Mean Profit per Trade

The ultimate measure of success for a prediction algorithm is it's ability to produce profit if applied to real trading. This profit can be approximated by applying the algorithm to the available data sets; the training set and the test set. As always, the performance on the training set may very well be subject to overtraining and should not be used as a final performance measure. The mean profit for a trading rule-oriented prediction algorithm is simple to compute. Just loop over the time interval in question and apply the trading rule to the computed prediction function f according to (14). Buys and sells are then executed according to the trading rule and accumulated for the whole time period.

The mean profit for a time series prediction is computed as

$$\frac{1}{N}\sum_{t=1}^{N} sign(\hat{y}(t) - y(t-1))\ (y(t) - y(t-1)), \tag{29}$$

i.e. a trade is assumed at every time step, in the direction of the predicted change.

A realistic evaluation of performance must also include trading cost in the performance calculation.

5 General Learning Issues

5.1 Statistical Inference

Model-free estimation and non-parametric statistical inference deal with the problem of finding a hypothesis function using a set of examples as primary information. The estimators can take many forms: algebraic or trigonometric polynomials, neural networks, step wise linear regression etc. Many of these techniques have been shown to share the common property of being *universal approximators*, capable of approximating any continuous function. In the case of neural networks it was first shown by Cybenko [8]. Note that these are theoretical results which may require a model with arbitrary complexity for convergence. The convergence also assumes an infinite example set. Furthermore, the data must be free of all noise. Conditions which are seldom fulfilled in real applications. The general limitations of statistical inference are also clearly understood and are formulated as the "bias/variance dilemma"[15].

5.2 Bias and Variance

To estimate an unknown function $f(\mathbf{P})$ that has produced a finite set of examples $(\mathbf{P}, \mathbf{T})$ the following general procedure for inductive learning can be applied:

1. Draw a random training set from the set of examples
2. Train one estimator $h_i(\mathbf{P})$ (e.g. a neural network)
3. Evaluate the estimator on a randomly picked test set of examples

Repeat steps 1-3 a large number of times, producing hypothesis functions $h_1, ..., h_N$.

Now concentrate on what we have produced for a specific point $\mathbf{p}$. We have a number of different estimators $h_1(\mathbf{p}), ..., h_N(\mathbf{p})$. Since they all depend on random selections of training data, they can themselves be viewed as outcomes of a random variable h_P with mean m_P and variance V_P.

The mean squared error for the predictions at the point $\mathbf{p}$ can be expressed as:

$$E_p = E[(h_P - f(\mathbf{p}))^2] = E[h_P^2 + f(\mathbf{p})^2 - 2h_P f(\mathbf{p})] = E[h_P^2] + f(\mathbf{p})^2 - 2f(\mathbf{p})E[h_P]. \quad (30)$$

Using the identity $V_p = E[h_p^2] - m_p^2$

$$E_p = V_p + m_p^2 + f(\mathbf{p})^2 - 2f(\mathbf{p})E[h_P] = V_p + (m_p - f(\mathbf{p}))^2. \quad (31)$$

The mean squared prediction error E_p thus is a random variable with two components termed *bias* and *variance*:

- Bias: $(m_p - f(\mathbf{p}))^2$. The amount by which the average estimator differs from the true value.
- Variance: V_p. The variation among the estimators.

The mean m_P can be estimated as $\sum_i h_i(\mathbf{p})/N$ and the variance V_P can be estimated as $\sum_i (h_i(\mathbf{p}) - m_P)^2/(N-1)$. Note that explicit estimation of E_P requires knowledge of the underlying function f. This is seldom, if ever, the case, otherwise there would be no need to estimate it. There exist statistical methods to estimate E_P without this knowledge of f. For example, the jackknife, and the bootstrap techniques [9]. However, it is possible to proceed without explicit calculation of E_P by looking at how the estimators' complexity affects the distribution for E_P. Depending on the estimators' complexity, we get the following extreme cases:

- Too low complexity (e.g. a straight line or a neural network with few weights) The computed estimators $h_1(\mathbf{p}), ..., h_N(\mathbf{p})$ will produce roughly the same values since a low complexity model does not have the power to express minor differences between different samples of training data. Hence the variance V_P will be low. However, the bias will be high, since all of the estimators differ in the same way from the true value $f(\mathbf{P})$.
- Too high complexity (e.g. a neural network with several layers and many weights) The computed estimators $h_1(\mathbf{p}), ..., h_N(\mathbf{p})$ will train precisely to each training set, thereby producing high variance V_P for the estimators since each estimator operates on different random samples of training data. However, the bias will be low, since the mean value m_P computed as $\sum_i h_i(\mathbf{p})/N$ will be centered around the true value $f(\mathbf{P})$.

Now turn to the "real" situation where only one single estimate $h_k(\mathbf{P})$ has been produced. It is still an outcome of the random variable $h(\mathbf{P})$ with its associated mean and variance. So, from what distribution on the estimates do we prefer to pick the single estimate: one with low-variance/high-bias or one with low-bias/high-variance? The choice is partly controlled by the complexity of the model. Looking at Equation (31), the correct answer would be that neither of the alternatives is optimal in terms of giving a minimal prediction error E_p. The best choice is a trade-off between low bias and low variance. Since the complexity affects the entities in different directions, we are facing what is called the "bias/variance" dilemma. A large variance has to be accepted in order to keep the bias low and vice versa.

According to Casdagli and Weigend [7], the position taken by most statisticians is, "for reasons of conservatism", to favor low-variance/high-bias over low-bias/high-variance. This choice results in nonlinear models with relatively low complexity and few parameters. These models work fine if the underlying function is equally simple and the interesting behavior is mainly due to outside perturbations. For more complicated functions, models with higher complexity must be used to get acceptably low prediction error. The price to be paid for this additional expressiveness is higher variance.

In some cases a low complexity model can be designed using prior knowledge about the specific application of interest. In such cases the bias is "harmless" since it is directed towards the real function $f(\mathbf{P})$. The bias then can be brought down without increasing the variance. This approach can be applied even to black box models such as neural networks. In *weight sharing* several synapses (connections between nodes) are using the same weight. In *radial basis networks* the receptive field of the neurons in the hidden layer can be preset to reflect known properties in the function to be modeled. The general situations are described with the terms "Weak" and "Strong" modeling.

5.3 Weak and Strong Modeling

The terms weak and strong modeling refers to the degree to which a model is preconditioned to reflect the underlying process to be modeled. A weak model makes few assumptions on what the real process looks like, whereas a strong model makes many assumptions. The traditional choice in the natural sciences has been to prefer strong models, which are tightly connected to the actual process. The parameters in such models can often be given a "meaning" in terms of slopes, constants, thresholds etc. However, there are also examples of weak modeling. In Physics one sometimes model dynamic systems as fairly general nonlinear functions [7]. The classic ARMA models described in Section 3.1 are also examples of weak models with few domain-specific assumptions.

5.4 Overfitting and Underfitting

The term *overfitting* is used to denote the situation, where the selected model type has too high complexity with respect to the "bias/variance" trade-off. The term *underfitting* denotes situations where the model has too low complexity.

5.5 Overtraining

The term overtraining refers to a situation, where a model is fit too closely to the training data, thereby decreasing the computed models generalization performance. The problem is caused by allowing the learning algorithm to keep iterating in an attempt to bring the total prediction error on the training data to an absolute minimum. However, when the prediction error has reached a point below a certain, problem specific, limit, the algorithm will start fitting properties in the training data which are not general but rather to be considered as noise. The generalization performance will therefor decrease from that point on in the learning process. The phenomena is normally connected to "weak modeling," where the model is totally unbiased and capable of fitting data from any data source. It is also connected to the issue of model complexity, since the point where the overtraining starts depends on the model's expressiveness. A model with a low complexity (e.g. a straight line) will be less sensitive to the problems with overtraining. A high noise level in data also increases the risk of overtraining, since noisy data gets interpreted more easily as real data by the training process. The risk for overtraining is also affected by the amount of data used for the training of the model. A lot of data prevents a model of a certain complexity from interpreting noisy data as real.

Statements such as "In the statistical context there is no such thing as overtraining"[45] are valid only in situations with low noise levels, combined with either a strong model or a weak one with low complexity. In other situations, the issue of overtraining is a reality and the training process, in our opinion, should be regarded as part of the model selection rather than a mere parameter estimation problem. Further support for this viewpoint is presented in Section 5.7.1.

The problem with overtraining can be solved either indirectly, by reducing the model complexity, or directly, by interrupting the learning algorithm. Refer to Section 5.7 for a further discussion on this issue.

5.6 Measuring Generalization Ability

The crucial measure for all learning algorithms is the ability to perform well when presented with unseen data (also called: out-of-sample performance). The difference between the performance on the training data and on unseen data is dramatically increased when low-bias models such as artificial neural networks are considered. The importance of good estimates of the performance on unseen data in such situations can be hardly overestimated. Apart from being the principal performance measure for a modeling problem, the generalization ability is also an important tool for model selection.

The theoretical aspects of why and when learning works is covered by *computational learning theory*, a field in the intersection of Artificial Intelligence and Computer Science. A result from the sub-field *PAC-learning (Probably Approximately Correct)* gives the following relation between the number of necessary examples m and the set of possible hypotheses H:

$$m \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right), \tag{32}$$

where ε is the error in a specific hypothesis and δ is the probability of a non correct hypothesis being consistent with all examples. Thus, by using at least m examples in the training, then with probability at least $1-\delta$ the produced hypothesis has an error of at most ε. Even if the practical results of PAC-learning are still limited, it centers the focus on two important things:

- The aim of learning is to find an approximately correct hypothesis. Traditional learning theory focused on the problem of *identification in the limit* where the hypothesis should match the true function exactly.
- The size $|H|$ of the hypothesis space, i.e. the model complexity, is a key issue for both estimation and control of generalization ability.

There exist a number of methods to estimate the generalization ability.

5.6.1 Test-Set Validation

The standard procedure for validation in most machine learning techniques is splitting the data into a training set and a test set. Sometimes the training set is further divided to extract a cross validation set (used to determine the stopping point to avoid overfitting). The test set is only used for the final estimation of the generalization performance. This approach is wasteful in terms of data, since not all the data can be used for training. The advantage is that no assumptions regarding error distribution or model linearity have to be made. In the case of neural networks, Weigend and LeBaron [40] have shown that the variation in results, due to how the splitting in the three sets is done, is much larger than the variation due to different network conditions, such as architecture and initial weights. The method used to show this is a variation of the procedure described in Section 5.2. By really generating a huge number of independent hypothesis functions the variance can be explicitly estimated. Weigend and LeBaron use predictions of traded volume on the New York Stock Exchange as their application. Volume data is known to contain more forecastable structures than typical price series. However, it is dominated by a noise component, which makes the analysis sensitive to the splitting of the data.

In our opinion this result should not be seen as a disqualification of the method of test-set-validation, but rather as an illustration of the general problem of all inductive inference methods. We investigate the statistical difficulties with predictions of noisy data further in Section 6.

5.6.2 Cross-Validation

Cross-validation takes the idea of test-set-validation to an extreme. Assume that the entire data set contains N data samples. One sample is left out and the remaining samples are used to train a model. The performance of this model is estimated by the squared error in the left out sample. The procedure is repeated for all N samples in the data set, thus producing N models with associated error estimates for the single left out point. The mean of these estimates is used as a total estimate of the prediction error for the model. If N is large, the method easily gets too expensive in terms of computations. Variations where more than one sample is removed from the data set were introduced in [14]. A method specifically developed for artificial neural networks was proposed in [29]. Instead of starting the training from scratch with each new training set, the weights from previous training are kept and used as starting values for the next model.

5.6.3 Algebraic Estimates

The various methods of cross-validation require the data to be split in separate sets for training and estimation of generalization error. This is often not possible to do when the total number of data points is very limited or the data is very noisy (a large training set is then necessary). However, a number of algebraic estimates of the generalization error exists, and they work without any splitting of the data. They all impose prior assumptions on the statistical error distribution. Some well known formulae are the Akaike Final Prediction Error (FPE)

$$FPE = MSE \cdot \left(\frac{1 + \frac{Q}{N}}{1 - \frac{Q}{N}} \right) \tag{33}$$

and the Generalized Cross Validation (GCV):

$$GCV = MSE \cdot \frac{1}{\left(1 - \frac{Q}{N}\right)^2} \tag{34}$$

The MSE is the average squared error, computed for the whole data set of N points. The methods work by penalizing the MSE with a term to compensate for the complexity of the model. A sufficiently powerful model with many weights can fit, as is well known, any data set. A low value of the MSE is therefor not sufficient to guarantee a low generalization error. The complexity is estimated by the number of *free parameters* Q.

In the case of neural network models, the value on Q is far from trivial! If the training is done with regularization such as early stopping, Tikhonov regularization or weight elimination, the number of free parameters is not the same as the number of weights!

5.7 Controlling Model Complexity

As described above, the model complexity is intimately connected to the "bias /variance dilemma". In powerful models such as neural networks with many weights, it's often necessary to impose some restrictions on the model in order to avoid too high model variance. The methods used are often termed "regularization" and include techniques such as Architecture Selection, Adding Noise to Data, Early Stopping and Tikhonov Regularization. Tikhonov *Regularization Theory* applied to neural network training algorithms is described in [11].

5.7.1 Architecture Selection in Neural Networks

The architecture of an ordinary feed-forward neural network is characterized by

- The number of hidden layers.
- The number of hidden nodes.
- The set of non-zero weights.

The problem of finding the best number of hidden nodes and the best set of non-zero weights can be approached in one of two ways:

- Network growth methods.

  The network is gradually increased in size by adding hidden nodes until the best performance is achieved. Such methods includes *structure-level adaptation* and *cascade correlation learning architecture* [12].

- Network pruning methods.

  The network is gradually decreased in size by either removing nodes (such as in *optimal brain surgeon OBS* [17] and *optimal brain damage OBD* [24]*),* by removing weights (such as in *weight-elimination* [43]) or by reducing weights (such as in *weight-decay*[21]. Haykin provides a survey of these common methods in [19]. Moody [28] presents some other common algorithms for control of the network complexity: the *Minimum Description Length (MDL)* [31] and *an information theoretic criterion* [3]. Both methods work by adding a complexity term to the ordinary mean squared error that is minimized in the training process.

Structure-Level Adaptation. In this method the network performance is monitored after the training phase has been completed. If the estimation error is larger than a desired value, a new node is added to the network, which is then trained again. If the weights connected to the inputs of a node fluctuate extensively between successive training it may be inferred that the node does not contribute to the function approximation and therefor should be removed.

Cascade Correlation Learning Architecture. The procedure starts with an empty hidden layer and an input and output layer determined by the specific problem at hand. New hidden nodes are then added one by one. Each new node gets weights connecting it to the input layer and to the existing hidden nodes. The new weights and some of the old ones are trained repeatedly when each new node is added. New nodes are added until satisfactory performance is attained.

Optimal Brain Surgeon (OBS) and Optimal Brain Damage (OBD). The basic idea of these methods is to identify the weights whose deletion from the network will cause the least increase in the value of the error function (normally the mean squared error). This is achieved by a quadratic approximation of the error surface. The methods are very computationally intensive and require inversion of the Hessian matrix for the network.

Weight-Decay. The idea of Weight-Decay is to force weights to take values either close to zero or relatively large. Add to the usual cost function a term which is a scaled sum of all squared weights, and minimize this new sum in the training phase. For a neural network with K weights and N examples in the training set, minimize the cost function

$$\sum_{i=1}^{N} (\hat{y}(t) - y(t))^2 + \lambda \sum_{i=1}^{K} w_i^2. \tag{35}$$

The last term groups the weights of the network into two categories: those that have a large influence on the mean squared error and those that have little influence on it. Weights that have little effect on reducing the mean squared error will be penalized because of the last term. Therefor, their absolute values will take values close to zero. On the other hand, weights that have large effect on reducing the mean squared error will not be penalized, since the last term is balanced by a hopefully greater reduction in the first term. The value on λ is crucial for a successful use of the method.

Weight-Elimination. The idea behind Weight-Elimination is simple: Add to the usual cost function a term which estimates the number of significant parameters, and minimize this new sum in the training phase. For a neural network with K weights and N examples in the training set minimize

$$\sum_{i=1}^{N} (\hat{y}(t) - y(t))^2 + \lambda \sum_{i=1}^{K} \frac{w_i^2/w_0^2}{1 + w_i^2/w_0^2}. \tag{36}$$

The last term is an estimate of the number of significantly sized weights. "Significant" is defined by the choice of w_0. For $|w_i| \ll w_0$ the cost is close to zero. All significantly sized weights will result in an extra cost and will therefor be penalized when the training algorithm attempts to minimize the cost function. The choices of λ and w_0 are crucial for the method to work. Guidelines for selection of λ can be found in [43].

Early Stopping. Early stopping is a method that addresses the complexity issue by combining training and estimation of the generalization performance. It is normally used for neural networks but could be also applied to other types of inductive learning techniques. Weigend et al. [42] suggests setting part of the training data apart, introducing a cross-validation set. The learning is done using the training set data but is stopped at the point where the performance (e.g. mean squared error) on the cross-validation set has its minimum. Therefor, this performance measure for the cross validation set is computed in parallel with the ordinary training algorithm that attempts to minimize the mean squared error on the training set data. The weights in the epoch (iteration) where the mean squared error on the cross validation set has its minimum are regarded as optimal and selected for the model. The produced model is then applied to the test set for a final estimate of the generalization performance.

In [41] and [39] Weigend and Rumelhart perform an interesting analysis on how the effective dimension of the hidden units in a neural network is changed during back propagation training. The effective dimension is a measure of the complexity in the model that is represented by the network at each instance during training. It turns out that the effective dimension starts at almost zero and gradually increases during training. This provides a justification for the use of oversized neural networks and early stopping. The early stopping should therefor be viewed as an architecture selection tool rather than an obscure method that stops the regression before the actual minimum is reached.

6 Random Walk

Financial time series are often claimed to be a total or "near to" random walk. A random-walk process is normally defined as

$$y(t) = y(t-1) + a(t) \tag{37}$$

where y is the time series in question and a is an error term which has zero mean and whose values are independent of each other. The change $\Delta y(t) = y(t) - y(t-1)$ is thus $a(t)$ and is hence independent of previous changes $\Delta y(t-1), \Delta y(t-2)$.... It can also be shown that (37) implies that even higher conditional moments of Δy (e.g. the variance of Δy) are independent. This consequence has led to discussions about the validity of the random-walk hypothesis as a theoretical model of financial markets [LeRoy 1989] and to alternative formulations. One such formulation is the *martingale* which is defined as

$$\Delta y(t) = \frac{y(t) + D(t) - y(t-1)}{y(t)}, \tag{38}$$

where $D(t)$ is the dividend paid during period t. A stochastic process following (38) rules out the conditional dependence of Δy, but not any higher moments of Δy, as shown in Mills [27]. This is consistent with the observations that financial time series go through quiet periods and also periods of turbulence. As Mills points out, this can be modeled by a process in which successive conditional variances are autocorrelated but not with the more restrictive random walk.

In any case, predicting stock prices is generally accepted to be a very difficult task. The stock prices do behave very much like a random-walk process at least when you look at the hit rates generated by most methods in use. Of course, this has direct implications both on the prediction task, the evaluation task and the application task.

6.1 Predicting an Almost Random-Walk Series

The prediction task is impossible if the time series to predict is an absolute random walk. In such a case *any* algorithm will produce 50% hit rate in the long run. The best we can hope for is that the time series we attempt to predict has a "near to" random-walk behavior, but with limited predictability. Degrees of accuracy of 54% hit rate in the predictions are often reported as satisfying results for stock predictions [37, 5].

6.2 Evaluating Prediction Algorithms for Almost Random-walk Series

The purpose of evaluating a prediction algorithm is to produce an answer "Yes" or "No" as to whether the algorithm really has predictive power. The evaluation task is directly effected by the relatively low degrees of accuracy that can be expected, even from a "successful" model. Some simple statistical exercises show the situations that may arise when trying to evaluate a prediction algorithm.

6.2.1 Scenario 1:

Assume that we are predicting a stock time series consisting of equal numbers of moves up and down during one year of 250 trading days. This is a realistic assumption. The average up/(up+down) ratios for a large number of Swedish stocks are in [20] shown to be between 50% and 51%.

For each day apply a totally random prediction algorithm. The algorithm simply produces a "1" (predicted move up in stock price) and a "0"(otherwise) totally at random. The probability that x predictions are correct is given by

$$P(\textbf{hit rate} = x) \;=\; \binom{250}{x} 0.5^x * 0.5^{250-x}. \tag{39}$$

This means that $P(\textbf{hit rate} \leq x)$ follows the binomial distribution $\textbf{binom}(250, 0.5)$ and therefor

$$P(\textbf{hit rate} > x) \;=\; 1 - P(\textbf{hit rate} \leq x) = \; 1 - \textbf{binom cdf}(x, 250, 0.5). \tag{40}$$

Here **binom cdf** denotes the binomial cumulative distribution function. Insertion of $x = 0.54 * 250 = 135$ yields $P(\textbf{hit rate} > 135) \;= 0.092$ as the probability that the random prediction algorithm gives a hit rate higher than 54%. Thus we are running a 9% risk of classifying the random algorithm as a useful predictive model. This corresponds to what statisticians call a "Type II error", i.e. accepting a false hypothesis. Lowering the required hit rate limit to 52% hit rate would increase the risk for a Type II error to $1 - \textbf{binom cdf}(0.52 * 250, 250, 0.5) = 24.33$. Not a very advisable thing to do as it looks!

6.2.2 Scenario 2:

Assume that we are evaluating technical indicators that produce sell and buy signals at an average rate of once a week. We have selected one hundred different indicators and want to do a proper test and decide if any of them has a predictive power which we define to be a hit rate of $> 55\%$. For a test period of 10 years we get 500 predictions from each indicator. They are compared with the actual stock chart and hit rates for the indicators that are computed. What's the probability that a random indicator slips through this test?

As before, we can compute the probability that a purely random indicator produces x or fewer correct signals (hits) when applied to a stock: $P(\textbf{hit rate} \leq x)$ follows the binomial distribution $\textbf{binom}(500, 0.5)$ and thus

$$P(\textbf{hit rate} > x) \;=\; 1 - P(\textbf{hit rate} \leq x) = \; 1 - \textbf{binom cdf}(x, 500, 0.5). \tag{41}$$

We compute $P(\textbf{hit rate} > 0.55 * 500) = 0.0112$ as the probability that a random indicator gives a hit rate higher than 55%. But since we started off with 100 different indicator series we must calculate the probability that we falsely accept ANY of the 100 random indicators! This risk is $1 - (1 - 0.0112)^{100}$ which calculates to 68%. It should be noticed that there are many hundreds of suggested indicators and rules, which are claimed to really predict future stock prices. In the light of what we have just seen, the mere selection of one of these indicators based on its past performance is statistically totally unacceptable.

6.2.3 Why Do We Get These Results

The primary reason why we may get results like the ones shown above is that the limits set for accepted prediction hit rates are too low. Increasing them to e.g. 60% would give considerably safer results. However, the problem would then be to find prediction algorithms that really produce such high hit rates for the required test period. The reason why so many papers present hit rates in the region below 55% may be simply that the prediction task is impossible (at least with the reported method) and the only way to get results that seem to be significant is to keep the required hit rate on a level where even a random predictor would produce them!

Scenario 2 above also pinpoints another important issue to bear in mind, especially when selecting a best performing algorithm or technical indicator. The dramatic increase to 68% in Scenario 2 is due to the fact that the selection is done *in sample*. Given enough number of different indicators it would be possible to find one with **any** hit rate. It corresponds to overfitting a powerful model to a set of data points. The conclusion is that a test set of data points must be kept untouched during the *entire* parameter estimation and model selection process. Although this may sound trivial, it is in fact very difficult to fulfill completely. Model selection is in effect even after the test results have been published since good prediction results probably get more attention than bad ones. For a thorough analysis of related problems refer to [25].

6.3 Guidelines for Developing Prediction Algorithms

The predictions of time series with near random-walk behavior emphasize some issues that should be considered when developing a prediction system:

- Ensure high data quality since a few errors in the input data may very well cause misinterpretations of the results in either direction. Watch out for outliers in the data.

- Handle missing data correctly. When analyzing hundreds of time series with daily data for many years it is unavoidable to have missing data points. Besides being a problem in the actual inference machinery, the missing data may also introduce unwanted biases when analyzing the output results. The easy way of removing the missing data before doing the analysis is not always advisable, e.g. when a set of stocks is analyzed in parallel. The individual stock time series often have missing data at different locations, which will result in a rather mangled data set after removal of all points where any of the stock data is missing. A better approach is to make the algorithm accept missing data and to do as well as possible when parts of the input data are missing.

- When predicting one step ahead, make sure you are really doing that! "Off by one" errors are seldom more fatal than in this case.

6.4 Guidelines for Evaluating Prediction Algorithms

The near random-walk behavior also calls for extreme precaution when evaluating the outputs from a developed prediction system. As has been pointed out earlier, the risk of interpreting random fluctuations as results of a successful prediction algorithm is always present and is sometimes surprisingly high. We propose the following guidelines for the testing and evaluation of prediction algorithms that are parts of a trading system:

- Beware of the data-snooping problem. Ensure that *all* parameter tuning, data selection and algorithm selection are done *in sample*. **Do not involve the test data**. The final evaluation on the test data set should be the last operation in the whole development cycle [3]. Experience shows that this is as difficult as it is important! The outstanding solution is to add a final test of the algorithm on data that simply didn't exist at the time of the development. In this way a true estimation of the predictability is achieved.

- Test the predictions on as much data as possible; on many stocks and for long time periods.

- Divide the available time period into smaller periods (e.g. annual) and perform the evaluation on these smaller parts as well as on the total time period. The purpose is threefold:

    1. The statistical level of significance is increased by evaluating many samples instead of one grand total.
    2. It is common that financial prediction algorithms turn out to depend on global properties such as long time trends or cyclic behavior in the time series. By splitting the data, such biases can be revealed.
    3. The variance of the prediction accuracy is estimated. It is important to realize that a prediction system that produces a massive loss during its first year in operation will be most likely scrapped, probably along with the developing team.

- Test the prediction algorithm on random-walk time series. If you manage to predict random walk, there is either a bug in the program or a weak evaluation procedure. Two tests can be performed:

[3] The publication of new methods is very much a part of the development cycle (evaluation). To be strict, the only way to avoid data snooping would be to publish **all** proposed prediction algorithms, and to propose all tested ones...

1. Generate and test the same amount of random data as the available real data. Repeat the testing several times (with different random "seeds") and record the test performance. By observing the spread between the different runs and the results from the real data prediction, the significance of the latter can be estimated.
2. Test the algorithm on a huge random test data set. The prediction results should converge to zero predictability (zero profit and 50% hit rate) as the size of the test data set increases.

- Use proper benchmarks. If a highly sophisticated and complicated prediction algorithm does not out-perform the naive prediction method, the application (or publication) of the algorithm should be taken under consideration. Likewise, a total accumulated profit below the "buy and hold" profit implies that the algorithm is rather useless as a real trading tool.

References

[1] E. Acar and P. Lequeux. Dynamic strategies: A correlation study. In C. Dunis, editor, *Forecasting Financial Markets*, Financial Economics and Quantitative Analysis, pages 93–124. John Wiley & Sons, Chichester, England, 1996.

[2] S. B. Achelis. *Technical Analysis from A to Z.* Irwin Professional Publishing, Chicago, 2nd edition, 1995.

[3] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, AC-19(6):716–723, December 1974.

[4] G. Andersson. *Volatility Forecasting and Efficiency of the Swedish Call Options Market.* PhD thesis, Handelshögskolan vid Göteborgs Universitet, September 1995.

[5] D. J. E. Baestaens, W. M. van den Bergh, and H. Vaudrey. Market inefficiencies, technical trading and neural networks. In C. Dunis, editor, *Forecasting Financial Markets*, Financial Economics and Quantitative Analysis, pages 245–260. John Wiley & Sons, Chichester, England, 1996.

[6] C. Bishop. Training with noise is equivalent to Tikhonov regularisation. *Neural computation*, 7(1):108–116, 1995.

[7] M. C. Casdagli and A. S. Weigend. *Exploring the continuum Between Deterministic and Stochastic Modeling*, pages 347–369. Addison-Wesley, 1993.

[8] G. Cybenko. Approximation by superpositions of a sigmoidal function. Technical report, University of Illinois, 1988.

[9] B. Efron. The jackknife, the bootstrap and other resampling plans. *Society for Industrial and Applied Mathematics (SIAM)*, 38, 1982.

[10] J. Ellman. Finding structure in time. *Cognitive Science*, pages 179–211, 1990.

[11] J. Eriksson, M. Gulliksson, P. Lindström, and P.-Å. Wedin. Regularization tools for training large feed-forward neural networks using automatic differentiation. *Optimization Methods and Software*, 10(1):to appear, 1998.

[12] S. E. Fahlman and C. Lebiere. *The cascade-correlation learning architecture.* Morgan Kaufmann, 1990.

[13] E. F. Fama. Efficient capital markets: II. *The Journal of Finance*, 46(5):1575–1617, December 1991.

[14] S. Geisser. The predictive sampling reuse method with applications. *Journal of The American Statistical Association*, 1975.

[15] Geman.S, E. Bienstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural Computation*, 5:1–58, 1992.

[16] N. A. Gershenfeld and A. S. Weigend. The future of time series: Learning and understanding. In A. S. Weigend and N. A. Gershenfeld, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past. Proceedings of the NATO Advanced Research Workshop on Comparative Time Series Analysis*, volume XV. Addison-Wesley, 1993.

[17] B. Hassibi, D. G. Stork, and G. J. Wolff. Optimal brain surgean and general network pruning. In *IEEE International Conference on Neural Networks, San Diego, Vol 1*, pages 293–299, San Fransisco, CA, 1993.

[18] G. Hawawini and D. B. Keim. On the predictability of common stock returns: Worldwide evidence. In R. A. Jarrow, V. Maksimovic, and W. T. Ziemba, editors, *Handbooks in Operations Research and Management Science*, volume 9: Finance, chapter 17, pages 497–544. North-Holland, Amsterdam, The Netherlands, 1995.

[19] S. Haykin. *Neural Networks, a comprehensive foundation.* Prentice-Hall, Inc, New Jersey, USA, 1994.

[20] T. Hellström and K. Holmström. Predicting the Stock Market. Research and Reports Opuscula ISRN HEV-BIB-OP–26-SE, Mälardalen University, Västerås, Sweden, 1998.

[21] G. E. Hinton. Connectionist learning procedures. *Artificial Intelligence*, 40:185–234, 1989.

[22] K. Holmström. Technical analysis models for the prediction of financial time series. Technical Report IMa-TOM-1997-10, Department of Mathematics and Physics, Mälardalen University, Sweden, 1997.

[23] B. LeBaron. Technical trading rules and regime shifts in foreign exchange. Technical report, Department of Economics, University of Wisconsin - Madison, Madison, WI, October 1991.

[24] Y. LeCun, J. S. Denker, and S. A. Solla. Optimal brain damage. *Advances in Neural Information Processing Systems*, 2:598–605, 1990.

[25] A. W. Lo. Data snooping and other selection biases in financial econometrics. In *Tutorials NNCM-96 International Conference, Neural Networks in the Capital Market Pasadena.*, 1996.

[26] D. Lowe and A. R. Webb. *Time series prediction by adaptive networks: A dynamical systems perspective.* IEEE Computer Society Press, 1991.

[27] T. C. Mills. *The Econometric Modelling of Financial Time Series.* Cambridge University Press, 1993.

[28] J. Moody. Prediction risk and architecture selection for neural networks. In V. Cherkassky, J. H. Friedman, and H. Wechsler, editors, *From Statistics to Neural Networks: Theory and Pattern Recognition Applications*, NATO ASI Series F. Springer-Verlag, 1994.

[29] J. Moody and J. Utans. Architecture selection strategies for neural networks: Application to corporate bond rating prediction. In A.-P. Refenes, editor, *Neural Networks in the Capital Markets*, pages 277–300. John Wiley & Sons, 1995.

[30] A.-P. Refenes. Testing strategies and metrics. In A.-P. Refenes, editor, *Neural Networks in the Capital Markets*, chapter 5, pages 67–76. John Wiley & Sons, Chichester, England, 1995.

[31] J. Rissanen. Modelling by shortest data description. *Automatica*, 14:465–471, 1978.

[32] P. M. Robinsson. Asymptotically efficient estimation in the precence of heteroskedacticity of unknown form. *Econometrica*, 55:875–891, 1987.

[33] C. Siriopoulos, R. N. Markellos, and K. Sirlantzis. Applications of artificial neural networks in emerging financial markets. In A.-P. Refenes, Y. Abu-Mostafa, J. Moody, and A. Welgend, editors, *Neural Networks in Financial Engineering, Proc. of the 3rd Int. Conf. on Neural Networks in the Capital Markets*, Progress in Neural Processing, pages 284–302, Singapore, 1996. World Scientific.

[34] R. J. Sweeney and P. Surajaras. The stability and speculative profits in the foreign exchanges. In R. Guiramares, B.Kingsman, and S.J.Taylor, editors, *A Reappraisal of the Efficiency of Financial Markets.* Springer-Verlag Heidelberg, 1989.

[35] F. Takens. Detecting strange attractors in fluid turbulence. In D. Rand and L.-S. Young, editors, *Dynamical Systems and Turbulence.* Springer, 1981.

[36] R. R. Trippi and J. K. Lee. *Artificial Intelligence in Finance and Investing.* Irwin Professional Publishing, Chicago, London, Singapore, revised edition, 1996.

[37] G. Tsibouris and M. Zeidenberg. Testing the efficent markets hypothesis with gradient descent algorithms. In A.-P. Refenes, editor, *Neural Networks in the Capital Markets*, chapter 8, pages 127–136. John Wiley & Sons, Chichester, England, 1995.

[38] E. A. Wan. Time series prediction by using a connectionist network with internal delay lines. In A. Weigend and N. Gershenfeld, editors, *Time Series Prediction. Forecasting the Future and Understanding the Past*, SFI Series in the Sciences of Complexity, Proc. Addison-Wesley, 1994.

[39] A. S. Weigend. On overfitting and the effective number of hidden units. In *Proceedings in the 1993 Connectionist Models Summer School*, pages 335–342. Hillsdale, NJ: Lawrence Erlbaum Associates, 1994.

[40] A. S. Weigend and B. LeBaron. Evaluating neural network network predictors by bootstrapping. Technical Report CU-CS-725-94, Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder, CO, May 1994.

[41] A. S. Weigend and D. E. Rumelhart. Weight elimination and effective network size. pages 457–476. Cambridge, MA: MIT Press, 1994.

[42] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Predicting the future: a connectionist approach. *Internatioanl Journal of Neural Systems*, pages 193–209, 1990.

[43] A. S. Weigend, D. E. Rumelhart, and B. A. Huberman. Generalization by Weight-Elimination applied to Currency Exchange Rate Prediction. In *International Joint Conference on Neural Networks*, pages 837–841. IEEE, 1991.

[44] H. White. Economic prediction using neural networks: The case of IBM daily stock returns. In *IEEE International Conference on Neural Networks, San Diego*, pages 451–459, San Diego, 1988.

[45] H. White. Parametric statistical estimation with artificial neural networks. Technical report, Department of Economics and Institute for Neural Computation, University of California, San Diego, 1991.

[46] S. Yakowitz. Nearest-neighbour methods for time series analysis. *Journal of Time Series Analysis*, 8(2):235–247, 1987.