A Random Walk through the Stock Market

Thomas Hellström

Department of Computing Science
Umeå University
S-901 87 Umeå, Sweden
Licentiate Thesis, 1998

December 10, 1998

Contents

Abstract

This thesis deals with the well-known problem of prediction of stock prices. The discussion is almost entirely confined to technical analysis, i.e. predictions are based on past price data only.

The material is divided into four parts. The first part is an introduction, where the near-random-walk behavior of the processes is illustrated. In the second part the prediction task is formalized. Existing benchmarks and testing metrics are surveyed, and some new measures are introduced.

The third part presents statistical investigations into the predictability of stock returns. The relevance of trends are tested, and a new rank measure is statistically examined for predictability.

Part four contains a survey of common techniques used for stock predictions, and also a discussion of general learning issues relevant to the prediction task. Some new tools are also presented. The stock trends are used as input variables in a k-nearest neighbor analysis, to find patterns of trend values that result in non-random future returns. The algorithm is extended with a selection procedure to find regions in the input space, where the future returns are asymmetrically distributed. The suggested algorithm is naturally applicable in areas other than stock prediction as well. It addresses situations, in which the overall predictability is low and can only be expected to apply in indeterminate regions of the input space.

The last Chapter of the thesis describes the principles behind, and the implementation of **ASTA**, an Artificial Stock Trading Agent written in the MATLAB language. The primary purpose of the project is to provide an easy-to-use environment for evaluation and development of multi-stock trading algorithms. The **ASTA** system has been applied successfully to historical stock data, and results covering 11 years of the Swedish stock market are presented.

Acknowledgments

Who finds the money when you pay the rent
Did you think that money was heaven sent
Lady Madonna, Lennon/McCartney

I wish to thank my supervisors, Professor Patrik Eklund and Dr. Kenneth Holmström. Patrik for having the guts to enroll a research student older than himself, and Kenneth for finding the financial resources. Patrik introduced me to the field of computational intelligence and has been encouraging all the way. Kenneth and I have been obsessed by the idea of computerizing our stock trading for many years. Since we started to work actively with the problem about three years ago, we have investigated most old and new techniques suggested for the task. Kenneth focused on the mathematical methods, while I turned my interest to the data mining aspects, and the use of machine learning techniques. Kenneth is the co-author of four of the papers included in this thesis.

I would also like to thank all the lovely people who helped me with proofreading. Any remaining errors are, as always, the author's responsibility. Complaints about non-remaining errors should, however, be addressed directly to Thomas Johansson, who obviously managed to read the whole thesis and find many logical errors (to the extent an error can be logical). Besides lifting the syntactical level of my work (he skips all formulas...), he also lifted the total number of readers to above 1 (author included), which is higher than the average academic level.

Since the subject of the thesis is closely related to money, it feels urgent to say that the financial support for my work has been of crucial importance for its realization. I'm therefore greatly thankful to the Applied Optimization and Modeling project, Department of Mathematics and Physics, Mälardalens University for having financed the majority of the work laid down in this thesis.

Umeå, Dec 10 1998
Thomas Hellström

Preface

In recent years the application of techniques, such as machine learning, data mining, knowledge discovery, and nonlinear optimization to financial prediction problems, has seen exciting results. Several conferences are totally devoted to the subject, including CF (Computational Finance) since 1993, CIFEr (Computational Intelligence in Financial Engineering) since 1995. Furthermore, most major conferences in the various fields of machine learning, and also in fields such as mathematical programming, have workshops and sessions dealing with methodologies for financial predictions.

This thesis investigates the special case of price forecasting for stocks. The discussion is almost entirely confined to technical analysis, i.e. predictions are based on past price data only.

The thesis is divided into four parts dealing with various aspects of the stock prediction problem. In the first part, problems of performance evaluation of near-random-walk processes are illustrated with examples, along with guidelines for avoiding the risk of data-snooping.

The second part describes the various approaches of technical and fundamental analysis and the prediction problem is formulated as a special case of inductive learning. 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. The third part presents statistical investigations regarding the predictability of stock returns. The examined data covers more than 200 stocks on the Swedish stock market for the time period 1987-1996. The results show a weak trend behavior and autocorrelation values that are stable even when the entire time interval is broken down to yearly intervals. A proposed concept of daily returns R_D and a comparison with the ordinary step returns R_S, show a significant difference in the data, caused by the holiday effect. It is also shown that seasonal variables, such as the month of the year, affect the stock returns more than the average daily changes. This is consequential for all methods, where the seasonal variables are not taken into account in predicting daily stock returns.

The fourth part surveys some of the standard methods used for predictions of stock prices, and also general learning issues, such as overfitting, bias/variance and generalization ability. Some new methods and tools are also presented. The stock trends are used as input variables in a k-nearest neighbor analysis, to find patterns of trend values that result in non-random future returns. The k-nearest neighbor algorithm is extended with a selection procedure to find regions in the input space where the future returns are asymmetrically distributed. The new algorithm is successfully tested on artificial stock data, and with trending patterns are introduced. The algorithm is applied to a number of national stock indexes as well: the American

Dow Jones, the German DAX, the British FTSE, and the Swedish Generalindex. The results are positive for some of the tested indexes and negative for others. The suggested algorithm is naturally applicable to areas other than stock prediction as well. It addresses situations, in which the overall predictability is low and can only be expected to apply in indeterminate regions of the input space.

The last Chapter of the thesis describes the principles behind and the implementation of **ASTA**, an Artificial Stock Trading Agent written in the MATLAB language. The primary purpose of the project is to provide an easy-to-use environment for developing multi-stock trading algorithms. A key ingredient in the system is a stable and realistic test bench, highlighting and helping to avoid the huge risks of data snooping involved in this kind of financial prediction. The behavior of the agent is controlled by buy and sell rules, which can be composed interactively or written as user-defined functions. Various types of combinations of rules and data screening can be easily performed, all within the MATLAB language syntax.

In addition to being a Windows-based development tool and test bench, the system can run in batch mode, where a supplied objective function maps a trading strategy to a profit measure. The batch mode can be used to tune parameters, or automate the development of trading strategies, for example with genetic methods. Examples of tuning parameters in standard technical indicators are presented. A modified Sharpe Ratio is used as performance measure.

To improve the performance of a given algorithm, the data from the simulated trades can be output and post-processed by classification methods, such as artificial neural networks or fuzzy rule bases. The **ASTA** system has been applied successfully to historical stock data, and results covering 11 years of the Swedish stock market are presented.

The content of the thesis is a further development of work reported in the following papers:

1. T. Hellström and K. Holmström. *Predicting the Stock Market.* Opuscula ISRN HEV-BIB-OP–26-SE, Mälardalen University, Sweden. Presented at ML2000 (Workshop on Machine Learning) in Riga, Latvia. November 28, 1997.

2. T. Hellström and K. Holmström. *The Relevance of Trends for Prediction of Stock Returns.* Presented at ECML-98 (the 10th European Conference on Machine Learning) in Chemnitz, Germany. April 24, 1998.

3. T. Hellström and K. Holmström. *Predictable Patterns in Stock Returns.* August 6, 1998. Opuscula ISRN HEV-BIB-OP–30-SE, Mälardalen University, Sweden.

4. T. Hellström. *ASTA - a Test Bench and Development Tool for Trading Algorithms.* UMINF-98.12 ISSN-0348-0542. Presented at EUFIT'98 (6th European Congress on Intelligent Techniques & Soft Computing) in Aachen, Germany. September 7-10, 1998.

5. T. Hellström and K. Holmström. *Parameter Tuning in Trading Algorithms using ASTA.*
   Accepted for presentation at Computational Finance CF99 in New York, USA. January 6-8, 1999.

Part I

What's So Special About Stock Predictions?

Chapter 1

Introduction

Out of college, money spent
See no future, pay no rent
All the money's gone, nowhere to go
You Never Give Me Your Money, Lennon/McCartney

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 [16] 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, evaluation, and application tasks.

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 does not 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 Chapter 3.

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 include the new algorithm as a new tool adjoining all the previously suggested and used stock evaluation methods. The predictive power of the new algorithm then decreases, 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.

Chapter 2

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 Chapter 3, it is very likely even for a random strategy to *appear* successful, although it has been tested for a seemingly long period.

2.1 The Efficient Market Hypothesis

Roll me and call me the tumblin' dice
Tumblin' Dice, Jagger/Richards

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, both publicly and 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 [41], Lowe and Welsh [22], and has been for long the "official" viewpoint in the academic society. However, in recent years many published reports show, that the efficient market hypothesis is far

[1] A more exhaustive description of the relation between random walk and the correlation is provided in Chapter 3.

from correct. Fama [12] goes as far as stating, in a review, that the efficient market hypothesis surely must be false.

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

2.2 Traders Viewpoint

When the shit hits the fan,
I'll be sitting on the can
When the whip comes down,
I'll be running this town
When the Whip Comes Down, Jagger/Richards

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 so that they can make profits. 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 [19], and Sweeney and Surarjaras [33], 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 reacts quite differently than a floor day trader, when a stock suddenly drops in value. These differences in time perspectives 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 rely 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.

Chapter 3

Random Walk

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

$$y(t) = y(t-1) + a(t) \tag{3.1}$$

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 (3.1) 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 and to alternative formulations. One such formulation is the *martingale*, which is defined as:

$$\Delta y(t) = \frac{y(t) + D(t) - y(t-1)}{y(t)}, \tag{3.2}$$

where, $D(t)$ is the dividend paid during period t. A stochastic process following (3.2) rules out the conditional dependence of Δy, but not any higher moments of Δy, as shown in Mills [23]. This is consistent with the observations that financial time series go through quiet periods as well as 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 the hit rates generated by most prediction methods in use are viewed. Of course, this has direct implications on the prediction, evaluation, and application tasks.

3.1 Predicting an Almost Random-Walk Time Series

The prediction task is impossible, if the time series to predict is an absolute random walk. In such a case *any* algorithm for prediction of the sign of $\Delta y(t)$ produces a 50% hit rate in the long run. The best we can hope for is that the time series we attempt to predict have 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, 4].

3.2 Evaluating Prediction Algorithms for Almost Random-Walk Time 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 affected by the relatively low degree 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.

3.3 Scenario 1

Assume that we are doing one-day predictions of a stock time series consisting of equal numbers of daily 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 is shown to be between 50% and 51%, refer to Section 8.1.

A totally random prediction algorithm is applied for each day. 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:

$$P(\textbf{hit rate} = x) \ = \binom{250}{x} 0.5^x * 0.5^{250-x}. \tag{3.3}$$

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

$$P(\textbf{hit rate} > x) \ = \ 1 - P(\textbf{hit rate} \leq x) = \ 1 - \textbf{binom cdf}(x, 250, 0.5). \tag{3.4}$$

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 would result in 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, apparently.

3.4 Scenario 2

Assume that we are evaluating technical indicators that produce sell and buy signals once a week on average. We have selected one hundred different indicators and want to conduct a proper test, and decide if any of them has 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 to the actual stock chart, and then hit rates

are computed for the indicators. What is the probability that a random indicator would slip 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:

$$P(\textbf{hit rate} > x) \;=\; 1 - P(\textbf{hit rate} \leq x) = \; 1 - \textbf{binom cdf}(x, 500, 0.5). \quad (3.5)$$

We compute $P(\textbf{hit rate} > 0.55 * 500) = 0.0112$ as the probability that a random indicator would produce a hit rate higher than 55%. But since we started off with 100 different indicators, 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, 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, can be statistically totally unacceptable.

3.5 Why Do We Get these Results

The primary reason for obtaining results like the ones shown above, is that the limits set for accepted prediction hit rates are too low. Increasing them to, say 60% would provide considerably safer results. However, then the problem would be to find prediction algorithms that really produce such high hit rates for the required test period. The reason that 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 obtain 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, pinpoints also another important issue to be borne in mind, especially when selecting the 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 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 quite 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 [21].

Part II

Problem Formulation

Chapter 4

Available Data

A prediction algorithm uses a set of known entities to produce a prediction of future values for 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 one used by the technical analysts. Their prognoses are based on past stock data only. The fundamental analysts include, in addition to the pure technical data, 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.

4.1 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; the 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; the 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 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, making it difficult to create a valid model for a longer period of time.
- y for different stocks may easily differ over several decades, and therefore can not be used as the same type of input in a model.

Therefore the *returns*, defined in (4.1), are often used instead.

4.2 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 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 hint at 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 [1]):

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, extracted from the company's financial statements. From these a number of useful variables can be calculated:

- p/e ratio. 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 the total sales.
- Debt ratio; liabilities divided by the 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, plenty of high quality data is published weekly in several financial magazines. In any case, the availability of huge amounts of data in machine readable form is much better for technical data than for fundamental data.

4.3 Derived Data

4.3.1 Return

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

$$R(t) = 100 \cdot \frac{y(t) - y(t-1)}{y(t-1)}. \tag{4.1}$$

A common variant is the log-return

$$R(t) = \log \frac{y(t)}{y(t-1)}. \tag{4.2}$$

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 $\ln(\frac{a}{b}) \sim \frac{a}{b} - 1 = \frac{a-b}{b}$.

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

$$R_k(t) = 100 \cdot \frac{y(t) - y(t-k)}{y(t-k)}. \tag{4.3}$$

4.3.2 Trend

The return $R_k(t)$ can be interpreted as the k-day price trend for the stock. It is convenient to divide by k in order to get the daily increase in price. Trend values for different values of k can then be analyzed on an equal basis. We suggest the following definition for the k-trend $T_k(t)$:

$$T_k(t) = \frac{100}{k} \cdot \frac{y(t) - y(t-k)}{y(t-k)}. \tag{4.4}$$

4.3.3 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 [2]. The standard definition is:

$$V = \sqrt{\frac{1}{N-1} \sum_{t=1}^{N} \left(\ln \left(\frac{y(t)}{y(t-1)} \right) - m \right)^2} \tag{4.5}$$

where m is the mean value,

$$m = \frac{1}{N} \sum_{t=1}^{N} \ln \left(\frac{y(t)}{y(t-1)} \right). \tag{4.6}$$

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 [32]. 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.

4.3.4 Volume increase

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

4.3.5 Turning Points

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

4.3.6 Technical Indicators

Technical stock analysis deals extensively with derived entities. For a thorough description of the most common technical indicators, see [1].

4.3.7 Artificial Data

Generating artificial data by adding noise to y (or by shifting data slightly) introduces a degree of time invariance. It is a way of telling the modeling algorithm 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.

4.3.8 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. One multi-stock approach is to predict the relative levels or a ranking list of a given set of stocks. The rank concept is introduced formally in Chapter 11.

4.4 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:

4.4.1 Dimensionality Reduction

It is advantageous to have as few input variables as possible when modeling the stock prices. It is therefore sometimes useful to compute aggregated entities that express many points in the time series into one value. These entities then can be used as the actual inputs in the modeling. Examples are running means of y, y_H, y_L and V.

4.4.2 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 neural network software packages.

2. Reduction of size differences over time in an input time series, which 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 vary by several hundred percent, if measured for some years. Therefore, 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 (4.1).
   - Transformation to log-returns as defined in (4.2).
   - Normalization using the mean.
   - Normalization using both the mean and the standard deviation.

As explained in Section 4.3, the first two measures are quite 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 frequent in time series predictions of financial data.

Normalization using the mean is defined as:

$$y_n(t) = y(t)/y_{mean}(t) \tag{4.7}$$

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:

$$y_n(t) = (y(t) - y_{mean}(t))/\sigma_y(t) \tag{4.8}$$

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

$$V_n(t) = (V(t) - V_{mean}(t))/\sigma_V(t) \tag{4.9}$$

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 common backward window size is $n_V = 30$ days. $V_n(t)$ expresses the number of standard deviations the volume figure differs from its running mean.

4.4.3 Linearization

It is often preferable to remove obvious nonlinear factors, which would otherwise have to be taken care of in the modeling. One example is the volume $V(t)$, 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:

$$V^{'}(t) = \frac{1}{1 + e^{-V(t)}} \tag{4.10}$$

4.4.4 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 a way that the variances along each of the new p dimensions are sorted by their sizes in a descending order. Therefore, 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 [17].

Chapter 5

Defining the Prediction Task

As the present now will later be past,
The order is rapidly fadin'
The Times they are A-changin', B. Dylan

A prediction problem involving stocks can deal with things other than pure predictions of price levels or price changes. It is sometimes of interest to make predictions of future traded volumes or of future volatility measures. A new interesting approach, in which the *rank* of the stocks is predicted, is presented in Chapter 11. However, this thesis mainly deals with predictions of stock price levels, or predictions of the returns $R_k(t)$ as defined in (4.3) or (4.2).

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

1. $R(t)$ has a relatively constant range even if data is being used as input for many years. 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 easier to evaluate a prediction algorithm for $R_k(t)$, by computing the prediction accuracy (hit rate) of the sign of $R_k(t)$.

A prediction problem for $R_k(t)$ or $y(t)$ can be formulated in many different ways. We investigate two major types:

- Fixed prediction-horizon methods

- Trading simulation methods

The former approach includes building models that attempt to predict the values a fixed number of days ahead. The latter approach simulates a real trading situation, where stocks, possibly many in parallel, are allowed to issue "buy signals" and "sell signals" according to a trading rule that corresponds to the model in the fixed horizon case.

A general problem with a fixed prediction horizon is that it does not reflect the way 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 into the future is as good as the same increase 15 days into the future. And the most commonly used 1-day horizon definitely does not correspond to a realistic way for stock trading.

A general problem with methods based on trading rules is that of statistical significance. The trading rule normally gives Buy or Sell signal only for a minor part of the points in the time series. While being one of the big advantages, it also presents serious statistical problems when computing levels of significance for the produced performance. It is easy to find a trading rule that historically outperforms any stock index, as long as it does not have to produce more than a few signals. It is clear that the performance measures for a trading rule that produces many signals for a given time interval, is more reliable than one that just produces a few, even if the profit is the same. However, it is *not* clear how this trade-off between profit and number of signals should be settled in an optimal way. The selection bias when choosing "good" trading rules, renders a complexity argument worthless, since even a very simple trading rule can produce very high profit if it has to signal just a few times. The general problem is left unsolved in the present work, with the single but important observation, that the number of produced signals should be as high as possible to render the trading rule credible. The trading system approach is examined further in Chapter 16.1.

5.1 Fixed Horizon Methods

Time waits for no one,
And it won't wait for me
Time Waits for No One, Jagger/Richards

The methods with a fixed prediction horizon can be further subdivided into what I call "The Time Series Approach" and "The Trading Rule Approach". Both can be described as special cases of general inductive learning.

5.1.1 Prediction as Inductive Learning

A general prediction task may be formulated as a special case of inductive learning. We define a time order and pick examples from time series as follows:

Given a set of N examples, $\{(\mathbf{X}(t), z(t+h)), t = 1, ..., N\}$, where $z(t+h) = f(\mathbf{X}(t)), \forall t$, return a function $g(\mathbf{X}(t))$ that approximates $f(\mathbf{X}(t))$ in the sense that the norm of the error vector $E = (e_1, ..., e_N)$ is minimized. For simplicity, $g(\mathbf{X}(t))$ is denoted $g(t)$. Each e_t is defined as $e_t = e(g(t), z(t+h))$, where e is an arbitrary error function. Thus we are aiming at approximating $z(t+h)$ with $g(t)$, where h is the prediction horizon. The prediction error in time step t is a function e of example output $z(t+h)$, and the function value $g(t)$.

To be more concrete we have to specify the following entities:

- The "Input", i.e. the $\mathbf{X}(t)$ vector.
- The "Output", i.e. the $z(t+h)$ vector.
- The error function $e(g, z)$.

- The error norm, based on some vector norm, for computing $\|E\|$.
- Prior knowledge (bias) 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 "The Time Series Approach" and "The Trading Rule Approach" can be expressed as shown below.

5.1.2 The 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:

$$\mathbf{X}(t) = (y(t), y(t-1), ..., y(t-k+1)) \tag{5.1}$$

and

$$z(t+h) = y(t+h). \tag{5.2}$$

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 h-day returns according to (4.1) 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. I.e.:

$$e_t = g(t) - z(t+h). \tag{5.3}$$

The root mean square error (RMSE) is used as the vector norm:

$$\|E\| = \sqrt{\frac{1}{N}\sum_{t=1}^{N} e_t^2}. \tag{5.4}$$

The learning task is to find a function g that minimizes $\|E\|$. The bias for g can be chosen in many ways. Common choices are:

- A linear AR model:

$$g(t) = \sum_{m=1}^{k} a_m y(t-m-1) + a_0 \tag{5.5}$$

- A general nonlinear model implemented by a multi-layer-feed-forward neural network. A 1-hidden-layer net with linear output defines g as

$$g(t) = \sum_j \left(w_j \; h\left(\sum_{m=1}^{k} w_{m,j} y(t-m-1) + w_{0,j} \right) + w_0 \right) \tag{5.6}$$

where h is a nonlinear function, such as the sigmoid function $h(x) = \frac{1}{1+e^{-x}}$.

The learning comprises finding the parameter values which minimize the error norm (5.4).

It is most common to let the minimized RMSE measure (5.4) be the end point in the prediction task. However, in order to utilize the predictions, a decision-taking rule has to be created. A simple rule when predicting returns is:

$$D(t) = \begin{cases} \text{Buy} & : \text{ if } \ g(t) > \alpha \\ \text{Sell} & : \text{ if } \ g(t) < -\beta \\ \text{Do nothing} & : \text{ otherwise} \end{cases}, \tag{5.7}$$

where α and β are threshold parameters for buy and sell actions, depending on the predicted change in the stock price.

The RMSE measure (5.4) is not always ideal for predictions of financial time series for the following reasons:

1. The RMSE treats all predictions, small and large, as equal. This is not always appropriate. Prediction points that would never be used for actual trading (i.e. price changes too small to be interesting) may cause higher residuals at the other points of more interest, in order to minimize the global RMSE.

2. A small predicted change in price, which actually resulted in a much higher value, but in the same direction, will be penalized by the RMSE measure. A trader is normally happy in this case, at least if, say the small positive prediction was large enough to issue a buy signal.

3. Several papers report a poor correlation between the RMSE measure and the profit made by applying a prediction algorithm, e.g. Leitch & Tanner [15] and Bengio [5]. A strategy that separates the modeling from the decision-taking rule, such as the one in (5.7), is less optimal than modeling the decision taking directly [25]. Arguments 1 and 2 both offer some explanations to these results.

5.1.3 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. This approach can also deal with the disadvantages of the 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:

$$T(t) = \begin{cases} \text{Buy} & : \quad \text{if } g(t) = 1 \\ \text{Sell} & : \quad \text{if } g(t) = -1 \\ \text{Do nothing} & : \quad \text{if } g(t) = 0 \end{cases}. \tag{5.8}$$

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

The input part $\mathbf{X}$ of each example $(\mathbf{X}(t), z(t+h))$ has the same form as in (5.1):

$$\mathbf{X}(t) = (y(t), y(t-1), ..., y(t-k+1)) \tag{5.9}$$

where $y(t)$ typically is the h-day returns or the closing price for the stock.

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

$$z(t+h) = R_h(t+h) \tag{5.10}$$

where $R_h(t+h)$ is the h-step return defined in (4.1) computed h days ahead.

The prediction error e_t in each step t is somewhat different from the time series approach. One alternative, which aims at maximizing the hit rate for the trading rule, is:

$$e_t = \begin{cases} 1: & g(t) = 1 & \text{AND} & z(t+h) < 0 \\ 1: & g(t) = -1 & \text{AND} & z(t+h) > 0 \\ 0: & \text{otherwise} & & \end{cases} . \tag{5.11}$$

The model is hence penalized if the predicted return has a different sign than that of the actual return $z(t+h)$.

Another alternative, which aims at maximizing the theoretical profit for the trading rule, is:

$$e_t = \begin{cases} -z(t+h): & g(t) = 1 & \text{AND} & z(t+h) < 0 \\ z(t+h): & g(t) = -1 & \text{AND} & z(t+h) > 0 \\ -z(t+h): & g(t) = 1 & \text{AND} & z(t+h) > 0 \\ z(t+h): & g(t) = -1 & \text{AND} & z(t+h) < 0 \\ 0: & \text{otherwise} & & \end{cases} . \tag{5.12}$$

This error measure equals the loss caused if trading was performed according to the trading rule.

The mean error for the non-zero cases is used as the vector norm. I.e.:

$$\|E\| = \frac{\sum_{t=k}^{N-h} e_t}{\sum_{t=k}^{N-h} a_t} \tag{5.13}$$

where

$$a_t = \begin{cases} 1 & if\ e_t \neq 0 \\ 0 & otherwise \end{cases} . \tag{5.14}$$

As before, the learning task is to find a function g that minimizes $\|E\|$. The bias for g is normally very strong, as displayed in the following example:

Example:

Inputs: Lagged stock prices are used as input:

$$\mathbf{X}(t) = (y(t), y(t-1), ..., y(t-k+1)). \tag{5.15}$$

Outputs: The *5-day-return* computed at $t+5$:

$$z(t+5) = R_5(t+5). \tag{5.16}$$

The error function is chosen according to (5.11) and the error norm according to (5.13). The unknown function g is restricted according to:

$$g(t) = \begin{cases} 1: & \text{if } mav_S(t) > mav_L(t) \quad \text{AND} \quad mav_S(t-1) \leq mav_L(t-1) \\ -1: & \text{if } mav_S(t) < mav_L(t) \quad \text{AND} \quad mav_S(t-1) \geq mav_L(t-1) \\ 0: & \text{otherwise} \end{cases} \tag{5.17}$$

where $mav_k(t)$ is a moving average of length k. I.e.:

$$mav_k(t) = \frac{1}{k} \sum_{m=0}^{k-1} y(t-m) \tag{5.18}$$

The trading rule is illustrated in figure 5.1.

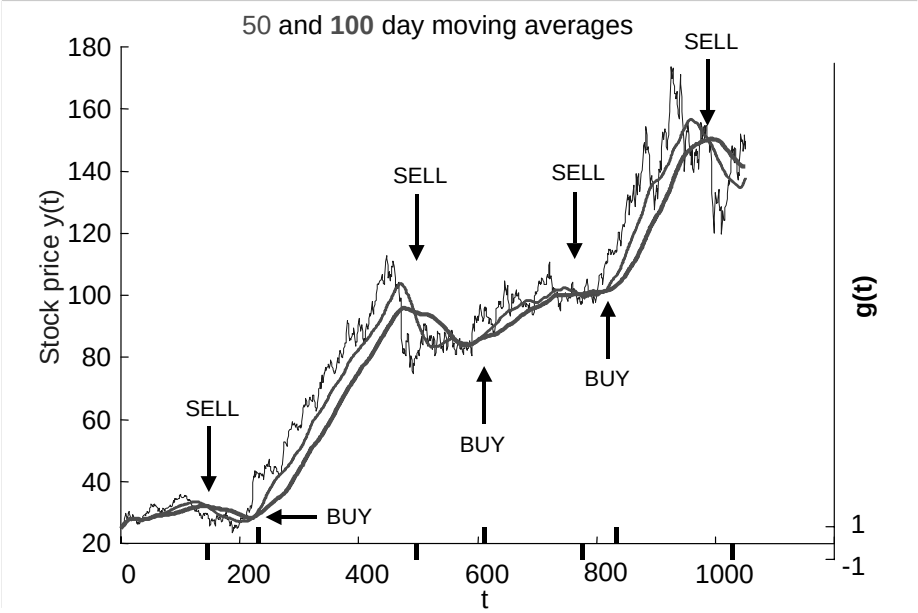

Figure 5.1: A trading rule based on moving averages

The learning in this example consists of finding optimal values to specify the function g, i.e.: the length variables in the moving averages S and L. Typical values are $S = 2$ and $L = 10$. The trading rule (5.8) signals "Buy," if the short moving average mav_S crosses the long moving average mav_L from below. A "Sell" signal is issued when mav_S crosses the mav_L from above. The optimal settings for S and L are determined by the learning process.

Note the difference between this and the Time Series Approach, where the prediction error at *all* points was added up to the mean error $\|E\|$. Also note, that $\|E\|$, viewed as a function of the parameters in g, may get a lot of local and even global minima when there are very few points where $e_t \neq 0$. These minima are clearly not what we are looking for when attempting to minimize $\|E\|$, and so they have to be avoided. It is necessary in these cases to regularize the problem by adding constraints to the number of predictions.

This formulation covers the so-called *technical indicators* that are very common in practical trading. Even if they are seldom described in this formal way, they can often be reformulated as a trading rule $T(t)$ as described above. The general trading

rule formulation can be also of interest when developing new prediction algorithms. The fact that the prediction error $g(t) - z(t + h)$ for the "Do nothing" situations is set to zero in the error measure e_t, points out the possibility to develop models that take into account the varying noise contents and predictability of the data-generating process. In this way, regions with low predictability produce more "Do nothing" cases.

5.2 Trading Simulation Methods

A trading simulation implements a trading-rule-based prediction algorithm in a system, where real trading is simulated as closely as possible. This means that the trading rule's Buy part initiates buy actions, and the Sell part initiates sell actions. In this way a more realistic situation is achieved than the previously described one of having a fixed horizon. Transaction costs for the trades can be also easily incorporated. A complete trading simulation includes much more than a pure prediction algorithm. Decision support is needed to decide *how many* stocks to buy and sell and, in the case of a multi-stock simulation, *which* stocks to prefer, if many stocks show a predicted increase in price. The trading simulation approach is examined further in Chapter 16.1.

5.3 Remarks

From the presented formalization of the prediction task we observe the following:

- The RMSE measure, commonly used in fixed-horizon prediction tasks, is just one of many choices, and by no means is "given by the gods" as a universal best alternative. The choice of error measure and error norm determines central properties like hit rate, profit, and risk compensation for the produced predictions.

- Modeling a trading rule by looking at fixed horizon values may be regarded, on good grounds, as inappropriate. A more natural method is a trading simulation, where both buy and sell signals are observed in relation to each other. The system **ASTA**, described in Chapter 16.1, supplies an environment where trading rules can be developed and evaluated in this fashion.

- I would like to point out, that the separation of the definition of the task and the performance measure is done purely to fit into the commonly used methods for prediction and evaluation. As described further on, these two concepts should be intertwined in order to reach an optimal solution of a prediction problem.

Chapter 6

Performance

How can I go forward when I don't know which way I'm facing?
How, J. Lennon

A way to measure the performance of a model is needed in two phases of the development cycle of the prediction system. First, during the modeling phase, where a model is selected or where optimal settings on unknown parameters in the model or in the trading rules have to be decided upon. Secondly, when the complete algorithm is put to test on historical data, to see if it serves the original goal of the development project. As was mentioned in Chapter 5.1.2, there are good reasons to use the same performance measure in these two phases. Otherwise, the final result may be less than optimal.

Evaluation of prediction performance is an important, difficult, and often overlooked stage in the development of financial prediction algorithms. In the case of an algorithm based on trading rules, one problem is the comparatively low number of produced trades, which constitutes a somewhat weak statistical basis for our performance measures. Apart from this, we have the problem with overtraining and selection bias. By tuning the parameters in the algorithm to maximize the performance on historical data, we always run a risk of fitting the algorithm too closely to the data set. Even for a rather "small" model (i.e. one with relatively few degrees of freedom), the problem must not be overlooked. The near-random-walk behavior in the data, combined with the comparatively low required prediction accuracy (a little higher than chance is normally regarded as sufficient,) make the situation very delicate. Illustrative examples were shown in Chapter 3.

A general problem with financial predictions is the non-stationary nature of the process, i.e. the performance of a trading strategy varies over time, due to changing global conditions affecting the market as a whole. One strategy may work fine in a trending market, while another works best in a non-trending market. Even if this should really be taken care of by the modeling phase, it often causes problems in the final evaluation made on historical data.

Before starting to optimize the behavior of our prediction algorithm, we must settle three issues:

- What performance do we want, or which performance metrics are relevant?
- What is the best way to measure the performance?

- To what do we compare our system's performance when we say that it is good? We need a good benchmark.

Let us start by investigating alternative performance measures.

6.1 Performance Metrics

In [28] Refenes provides a survey of a large number of measures of performance for financial predictions. In this section I present the ones I have found essential, as well as some new metrics believed to be necessary. Two properties are considered particularly important: relevance to the prediction task and the availability of a benchmark. The latter is extremely important, since we are dealing with (at least) near-random-walk processes, in which for a prediction system to be successful, it barely has to outperform pure chance.

6.1.1 The Time Series Approach

Let me forget about today until tomorrow
Mr. Tambourine Man, B. Dylan

The metrics for testing algorithms constructed according to the Time Series Approach work by comparing the predicted values to the actual outcome in the time series. 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 $\{y(t), t = 1, ..., N\}$. 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 $\{R(t), t = 1, ..., N\}$. We assume an h step horizon in the predictions. I.e.: The predictions $\hat{y}(t)$ and $\hat{R}(t)$ are produced at time $t - h$. The following metrics are relevant as performance measures for algorithms constructed according to the Time Series Approach.

Stock Price RMSE

The RMSE for the predicted stock prices y is defined as:

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=h}^{N}(y(t) - \hat{y}(t))^2}. \tag{6.1}$$

Return RMSE_R

The RMSE for the predicted returns R can be formulated correspondingly:

$$RMSE_R = \sqrt{\frac{1}{N}\sum_{t=h+1}^{N}(R(t) - \hat{R}(t))^2}. \tag{6.2}$$

Hit Rate H_R

The hit rate of a return predictor indicates how often the sign of the return 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. I.e.:

$$H_R = \frac{\left|\left\{t|R(t)\hat{R}(t) > 0, t = 1, ..., N\right\}\right|}{\left|\left\{t|R(t)\hat{R}(t) \neq 0, t = 1, ..., N\right\}\right|}. \tag{6.3}$$

The norm of the set in the definition is simply the number of elements in the set.

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:

$\hat{R}(t) = 0$	$R(t) > 0$
$\hat{R}(t) = 0$	$R(t) < 0$
$\hat{R}(t) = 0$	$R(t) = 0$
$\hat{R}(t) > 0$	$R(t) = 0$
$\hat{R}(t) < 0$	$R(t) = 0$

Regardless of the choice made for the classification of these situations, the result is invariably an asymmetric treatment of the positive and negative returns. Since the zero-valued one-day returns can account for more than 20% of the samples in typical stock data (see tables 8.2, 8.3,) they would result in "Up fractions" arbitrarily either greater 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 outcome, "Up fractions" very close to 50% are achieved. Therefore, in the case of one-day returns (i.e. $h = 1$), a hit rate H_R, significantly greater than 0.5, can be regarded as true predictions of the sign of the returns. Refer to Section 8.1 for further discussion.

Mean Profit per Trading Day

The ultimate measure of success for a prediction algorithm is its ability to produce profit if applied to real trading. This profit can be approximated for a Time Series Approach method, by assuming a trade at every time step, in the direction of the predicted change. The mean profit P per trading day for a time series prediction is computed as:

$$P = 100 \cdot \frac{1}{h} \cdot \frac{1}{N-h} \cdot \sum_{t=h+1}^{N} \frac{sign(\hat{y}(t) - y(t-h))\ (y(t) - y(t-h))}{y(t-h)} \tag{6.4}$$

where

$$sign(x) = \begin{cases} 1 & if\ x > 0 \\ -1 & if\ x < 0 \\ 0 & if\ x = 0 \end{cases}. \tag{6.5a}$$

6.1.2 The Trading Simulation Approach

The following metrics are relevant for algorithms constructed according to the Trading Simulation Approach.

Total Profit

An intuitively appealing and very common measure is the total wealth achieved by the trader, when simulating trading over the available training data period. The wealth is often presented as a function of time in a so-called *equity diagram*.

Average Profit per Day

To get around the cumulative effect in the total-profit measure (more about that in Section 6.4.2,) one could compute the average profit per day (percentage) and use that as a measure of the trading performance. By this we lose the possibility to evaluate the trader's behavior during the varying global market conditions that arise over any longer simulation period.

Profit per Year

Computing the annual profit achieved by applying the trading algorithm, would provide a compromise between using the total profit and the average profit per day. The mean annual profit could be used as total measure for the entire time period. However, it is also very important to pay attention to the performance in each individual year.

Statistical Significance

The profit from a Trading-Rule-based system has a very weak statistical significance, if the number of trades produced during the simulation is too low. The number of trades is therefore of central importance, and should be presented along with the trading results. Histograms, such as the one shown in figure 6.1, give a good overview of the annual performance.

6.2 What Is the Best Way to Measure the Performance?

For the loser now will be later to win
The times they are a-changin', B. Dylan

It is important to realize that the profit measure we select for optimization and evaluation is a stochastic variable, which does not have any single "value" that can be measured. Instead, it has a probability distribution that can be described by its statistical moments, such as the *expectation value* and the *variance*. What we get when we compute the annual profit for a year of trading simulation for example, is merely *one* sample from this distribution (assuming it is stationary over the years). The task of "maximizing" the profit by changing the parameters in the model now becomes less well-defined. Obviously, the common way of using the *mean* value is

just *one* option. Other choices could be the lower limit of a confidence interval, the *median value* or the *Sharpe Ratio.* The issue connected to trading simulation is further discussed in Section 16.6, where the profit is examined as a function of various parameters in the model. However, the situation is the same, regardless of the performance measure. An RMSE computed for one-day horizon predictions may, for example, vary considerably depending on the degree of stationarity of the process. It is therefore essential to present performance measures not only for the whole investigated time period, but also as annual values, for example. In this way, the spread in results and the dependencies on idiosyncrasies in the data become highlighted.

6.3 Benchmarks

Besides a performance measure, we need something to which to compare the performance. Whether it is a good or bad performance depends on the alternatives. Even a prediction algorithm that loses money sometimes, may turn out to be successful when compared to some of the alternatives.

The correct way to evaluate prediction performance depends on the approach chosen for the problem formulation. Suitable benchmarks for the Time Series Approach and the Trading Simulation Approach are described separately below.

6.3.1 The Time Series Approach

In this section the new benchmarks $\varepsilon - increase\ prediction$ and *Naive Prediction of Returns* are introduced as extensions to the well-known *Naive Prediction of Stock Prices.* The new measures T_r, HR_ϵ, HR_N, and P_r are suggested as testing metrics for performance evaluation relative to these benchmarks.

Naive Prediction of Stock Prices

The naive price predictor asserts that the best estimate of the following price $y(t+h)$ is today's price $y(t)$. This is a direct consequence of the random-walk hypothesis. It is always a good idea to measure the quality of a predictor in relation to this trivial predictor. This is done in the *Theil coefficient of inequality* T_y, defined as the quotient between the $RMSE$ for the investigated predictor and the $RMSE$ for the naive price predictor . I.e.:

$$T_y = \frac{RMSE_y}{RMSE_{yN}} = \frac{\sqrt{\sum_{t=h+1}^{N} (y(t) - \hat{y}(t))^2}}{\sqrt{\sum_{t=h+1}^{N} (y(t) - y(t-h))^2}}. \tag{6.6}$$

The Theil coefficient is often referred to as *the information coefficient* or the *t-test.* For $T_y > 1$ the predictor is worse than the naive price predictor, while $T_y < 1$ implies that the predictor is making better predictions.

$\varepsilon -$ increase Prediction

The $\epsilon - increase$ predictor asserts that the best estimate of the following price $y(t+h)$ is today's price $y(t)+\varepsilon$. The purpose of adding a small positive number ε to today's price is simply to enable computation of the hit rate for this predictor. The hit rate of this predictor provides a measure for the overall positive trend that "normally" makes an increase in price more likely than a decrease. The hit rate for the $\epsilon - increase$ predictor is defined as:

$$H_\epsilon = \frac{|\{t|R_h(t) > 0, t = h+1, ..., N\}|}{|\{t|R_h(t) \neq 0, t = h+1, ..., N\}|}. \tag{6.7}$$

The measure HR_ϵ is defined as the ratio between the predictor's hit rate H_R (as defined in (6.3)) and H_ϵ. I.e.:

$$HR_\epsilon = \frac{H_R}{H_\epsilon}. \tag{6.8}$$

HR_ϵ compares the hit rate of the predictor to that of the $\epsilon - increase$ predictor. For $HR_\epsilon < 1$ the predictor is worse than the $\epsilon - increase$ predictor, while $HR_\epsilon > 1$ implies that the predictor is making better predictions.

Naive Prediction of Returns

The Theil coefficient (6.6) 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 naive prediction of stock returns asserts today's return $R_h(t)$ (price increase since $t-h$) as the best estimate of $R_h(t+h)$. This naive prediction is formed from the observation of a one-step memory in the price generating process. In Section 8.2 the autocorrelation of stock returns is shown to exhibit a significant first lag component that indicates a correlation between adjacent returns. It is a good idea to measure the $RMSE$ of a return predictor in relation to the $RMSE$ for this naive return predictor. This is done in *Theil coefficient for returns* T_r defined as:

$$T_r = \frac{RMSE_r}{RMSE_{rN}} = \frac{\sqrt{\sum_{t=h+1}^{N} (R_h(t) - \hat{R}_h(t))^2}}{\sqrt{\sum_{t=h+1}^{N} (R_h(t) - R_h(t-h))^2}}. \tag{6.9}$$

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.

The Naive Prediction of Returns can also be compared on the basis of hit rate. This is the purpose of our suggested measure HR_N ("Hit rate relative to the Naive Predictor") and is described below. The hit rate H_N for the naive return predictor is first computed as:

$$H_N = \frac{|\{t|R_h(t)R_h(t-h) > 0, t = h+1, ..., N\}|}{|\{t|R_h(t)R_h(t-h) \neq 0, t = h+1, ..., N\}|}. \tag{6.10}$$

The Naive Prediction of Returns assumes, that an upswing is followed by yet another upswing the next day, and a downswing by yet another downswing.

The Relative Hit Rate HR_N is defined as the ratio between the hit rate of the predictor H_R (as defined in (6.3),) and that of the naive return predictor. I.e.:

$$HR_N = \frac{H_R}{H_N}. \tag{6.11}$$

HR_N compares the hit rate of the predictor relative to that of the naive return predictor. For $HR_N < 1$ the predictor is worse than the naive return predictor, while $HR_N > 1$ implies that the predictor is making better predictions.

Buy-and-Hold

The *Buy-and-Hold* return r_b for a time period $\{t...t+n\}$ of a stock is defined as:

$$r_b = 100 \cdot \frac{(y_{t+n} - y_t)}{y_t}, \tag{6.12}$$

I.e. the profit (expressed in percentage) made when buying at the start and selling at the end of the time period. The *Profit Relative to Buy-and-Hold* P_r is defined as the ratio between the predictor's accumulated *Mean Profit per Trading Day* P (as defined in (6.4),) and the *Buy-and-Hold* return. I.e.:

$$P_r = \frac{(1 + P/100)^n - 1}{r_b/100}. \tag{6.13}$$

For $P_r < 1$ the predictor's profit is worse than the Buy-and-Hold alternative, while $P_r > 1$ implies that the predictor is making higher profits.

6.3.2 The Trading Simulation Approach

The natural benchmark when doing Trading Simulation is some kind of stock index, which is also most often used by professional broker companies. A broker who outperforms the index gets bonuses, and the one who does not might risk his job. Mutual funds in particular are governed by these attitudes. Even if the individual traders don't lose their jobs, the managers know that their funds will lose customers, if they start doing worse than the global stock index. This is one of the reasons that so many mutual funds acting on one particular market perform almost identically.

For our purposes, however, it is a reasonable benchmark because it compares the performance to a very available alternative: that of buying a mutual fund instead of doing the trading ourselves.

The following two kinds of indexes have been tried:

- The Swedish index *Generalindex.*
- An artificial index, in which all the stocks available for the artificial trader are assigned equal parts.

No significant differences affecting the performance evaluation have been found between the two kinds of indexes, and therefore the official *Generalindex* has been chosen as a benchmark. It can be argued, that the change in the index underestimates the average profit achieved for the stocks that constitute that index. The reason for this, would be that the dividends for the stocks are not taken into account

in the calculation of index[1]. On the other hand, the dividends are not included in the profit calculations for the tested prediction algorithms either, which means, that it is subject to the same underestimation. Therefore, the change in index is considered a relevant benchmark when comparing different prediction methods[2].

6.4 Conclusions Regarding Performance Evaluation

The following presentation of results is recommended for a proper performance evaluation of prediction algorithms. General guidelines are also offered in Section 7.2.

6.4.1 The Time Series Approach

The results from a prediction system designed according to the Time Series Approach should be presented with annual figures for the following entities:

- If absolute prices y are predicted: the *Theil coefficient of inequality* T_y (or $RMSE_y$ and $RMSE_{yN}$ separately)
- If returns R are predicted: the *Theil coefficient for returns* T_r (or $RMSE_r$ and $RMSE_{rN}$ separately)
- The *hit rate relative to the* ϵ-increase predictor HR_ϵ (or H_R and H_ϵ separately)
- The *hit rate relative to the* naive increase predictor HR_N (or H_R and H_N separately)
- The *Profit Relative to Buy-and-Hold* P_r (or P and r_b separately)
- The number of predictions

In addition to the annual values, summary and mean values for the entire time period are also useful for fast comparison of methods.

6.4.2 The Trading Simulation Approach

The result of a trading system should be presented as annual profits together with the increase in index. The mean difference between these two figures constitutes the net performance of the system. The performance may be displayed in so-called equity diagrams, as shown in figure 6.1 (examples from the ASTA system).

A global stock index is presented in the same diagram for comparison. The curves are scaled, so the leftmost point has a wealth of 1 for both the trader and the index. The values for other points along the date axis can be then interpreted as wealth relative to the one at the starting point. A value 2.10 means, for example, that the start capital has grown to 210% of its original value. Therefore, the final

[1] For the Swedish stock market, this underestimation has been around 3% per year for the period 1988-1997

[2] Some official indexes, for example the Swedish *Affärsvärldens Index,* are automatically compensated for this effect.

value at the very right of the diagram is the net result after trading for the entire time period has been completed. The major drawback of this method is that the trades in the beginning of the time period affect the end result more than the ones in the end of the time period. This is a consequence of the cumulative nature of the simulation. The profits in the beginning of the time period are being reinvested, and therefore appear "several times" in the total wealth resulting from trading during the entire time period [3].

The performance can be also displayed in a tabular format, as shown in the table part of figure 16.2. The second last line of the table in figure 16.2 presents the annual profit to be above the index. The mean value of this entity is presented in the second rightmost column, and represents a one-figure performance measure. However, the individual figures for each year should also be considered. The Sharpe Ratio produces a risk-dependent measure, and is discussed further in Section 16.6. The number of trades for each year is also highly important for the statistical significance of the results.

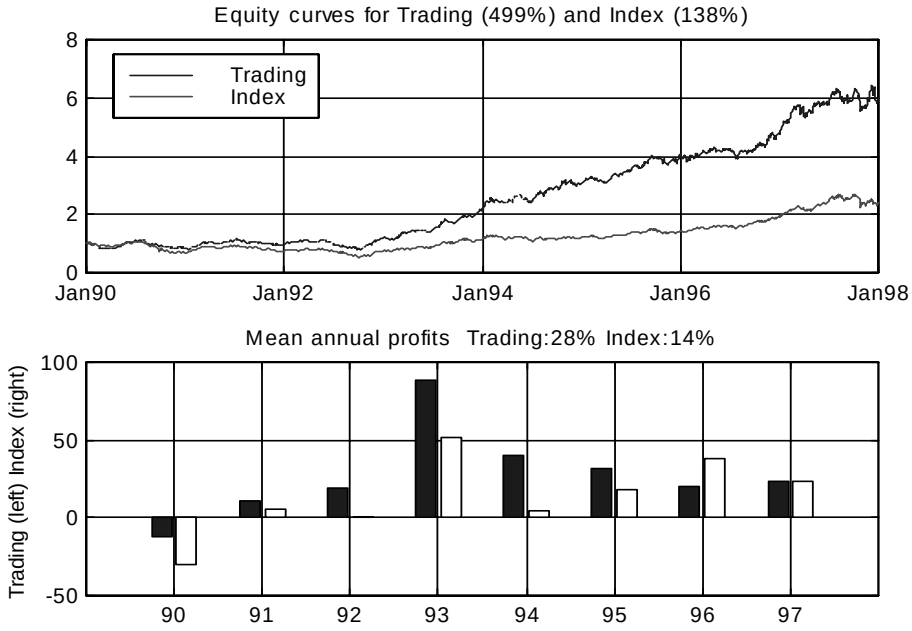

Figure 6.1: Presentation of performance for a (successful) trading strategy

[3] This is a nice effect if you want to show off your trading and stun people with an exponential increase in wealth. It is commonly used by the financial institutes when presenting results for their mutual funds over the years. However, the index curve is often not included in these presentations, since it would reveal that the mutual fund actually has not performed any better than the market in general. One has to realize that a constant economical growth of, say 5% over a period of 10 years, turns out a seemingly impressive exponential wealth curve for the average stock. With or without clever trading systems.

Chapter 7

Guidelines for Algorithms

The near-random-walk behavior of the stock-price times series calls for exerting extreme precaution, when developing and evaluating the outputs from any prediction system. In this chapter some guidelines are presented, for limiting the risk of data-snooping in the development and evaluation phases.

7.1 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, because 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 solving this situation is to remove the missing data before doing the analysis. This solution is not always advisable, for example, when a set of stocks is analyzed simultaneously. The individual stock time series often have missing data at different locations, which results 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 do as well as possible when parts of the input data are missing.
- When predicting one step ahead, make sure this is what you are really doing. "off-by-one" errors are seldom more fatal than in financial predictions.

7.2 Evaluating Prediction Algorithms

As pointed out earlier, the risk of interpreting random fluctuations as results of a successful prediction algorithms 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 [1]. 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 is most likely to be 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:

  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 outperform 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.

[1] 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 propose all tested ones...

Part III

Data Mining the Stock Market

Chapter 8

Basic statistics

It took me so long to find out, and I found out, Ah, ah, ah, ah, ah, ah
Day Tripper, Lennon/McCartney

This part presents results from a statistical analysis of stocks of the Swedish stock market for the period 1987-1996. The purpose of the project is to examine, in general terms, the statistical properties relevant to the development of prediction algorithms. Therefore, at this stage, we are not aiming at extracting patterns or dependencies that can be utilized directly for trading purposes.

Results for two sets of stocks are normally presented; *SXG* which is a compilation of 32 major stocks with active trading and *SXBIG* which is a compilation of 207 major and minor stocks (including those in *SXG*). The stocks included in *SXG* are listed in Table 8.1.

8.1 Returns

The k-step return $R_k(t)$ of a stock-price time series $y(t)$ was previously defined as:

$$R_k(t) = 100 \cdot \frac{y(t) - y(t-k)}{y(t-k)}. \tag{8.1}$$

The basic statistical properties of $R_k(t)$ for the two sets of stocks are listed in Tables 8.2 and 8.3. The values in the tables are mean values for the included stocks. Each column shows data for one particular value of k.

The last six lines in the tables show the distribution of signs for the returns. "$Return = 0$" is the fraction of returns equal to zero. "$Return > 0$" is the fraction of returns greater than zero and "$Return < 0$" is the fraction of returns less than zero. *"Up fraction"* is computed as:

$$100 \cdot \frac{\text{"}Return > 0\text{"}}{\text{"}Return > 0\text{"} + \text{"}Return < 0\text{"}}, \tag{8.2}$$

which is the positive fraction of all non-zero moves. *"Up fraction"* is a relevant measure, when it comes to evaluating the hit rate of prediction algorithms. Looking at one-step returns in the tables, the "$Up\ fraction$" for the *SXG* stocks is 50.8% and for the *SXBIG* stocks is 50.6%. The "$Mean\ Up$" and "$Mean\ Down$" columns show the mean value of the positive and negative returns respectively. The fractions of zero returns in the data material are somewhat surprisingly high, 14.1% for the

Table 8.1: Stocks in set SXG

	From	To
AGA A	870102	961230
ABB A	870102	961230
AssiDoman	940408	961230
Astra A	870102	961230
Atlas Copco A	870102	961230
Autoliv	940609	961230
Avesta Sheffield	870126	961230
Electrolux B	870102	961230
Ericsson B	870102	961230
Hennes & Mauritz	870102	961230
Incentive A	870102	961230
Industrivard A	870116	961230
Investor A	870102	961230
Kinnevik B	870105	961230
MoDo B	870102	961230
Pharm & Up SDB	871015	961230
SE-Banken A	870102	961230
Sandvik A	870102	961230
SCA B	870102	961230
SHB A	870102	961230
Skandia	870102	961230
Skanska B	870102	961230
SKF B	870102	961230
Sparbanken A	950609	961230
SSAB A	890703	961230
Stora A	870102	961230
Sydkraft C	870102	961230
Trelleborg B	870102	961230
Trygg-H SPP B	891208	961230
Volvo B	870102	961230
Allgon B	880527	961230
Nokia A	870102	961230

SXG stocks and 23.4% for the *SXBIG* stocks. The higher value in the latter set is related to the lower degree of activity in the smaller stocks included in *SXBIG*. The zero returns must be dealt with in a proper way when evaluating hit rates for prediction algorithms. The "*Up fraction*" circumvents the zero returns by simply removing them before calculating the hit rate. In this way, the zero returns are counted as both increases and decreases, in equal proportions. A similar procedure is proposed in section 6.1.1 for a test metric, when doing stock predictions.

8.2 Autocorrelation

Much of the literature about market efficiency and stock price predictability is focused on the properties of the autocorrelation function of the stock returns. Fama [12] surveys a number of investigations and concludes that "The new research produces precise evidence on the predictability of daily and weekly returns from past returns". In this section we present our own results from investigations of data from the Swedish stock market.

The autocorrelation for a time series X describes how well X is correlated to itself at two different times. For a stationary X the autocorrelation is defined as a function of the time difference τ as:

$$A_X(\tau) \;=\; \frac{Cov[X(t), X(t+\tau)]}{\sigma_X^2} \tag{8.3}$$

where

$$Cov[X(t), X(t+\tau)] \;=\; E[(X(t)X(t+\tau)] - \mu_X^2. \tag{8.4}$$

μ_X and σ_X are the mean and standard deviation respectively for the time series X. The E denotes the expected value of the argument.

The autocorrelation $A_X(\tau)$ can be viewed as a measure of the τ-step memory in the process generating X, computed as a mean value over the entire time series. By computing $A_X(\tau)$ for $\tau = 1, 2, 3, ...$, the autocorrelation can be plotted as a function of τ. It should be noticed, that only linear dependencies are fully revealed by the autocorrelation. However, it is often used as a general indicator of predictability, even for nonlinear functions.

We want to investigate the autocorrelation for the one-step stock returns $R_1(t)$ defined in (8.1). The autocorrelation for $R_1(t)$ (denoted ACF) is computed using (8.3).

Figure 8.1 shows the ACF for the *SXG* stocks. For each lag (value on τ), the mean value of the ACF for the 32 stocks is plotted. To test the stationary state assumption for the underlying process, the data is broken down to five equal blocks, each covering about two years of stock data. The graph to the left in Figure 8.1 shows the ACF for the entire time period. The graph to the right shows one curve for each two-year period.

The autocorrelation values are overall very low. The autocorrelation for lag 1 is clearly positive, whereas the following lags, 2 and 3, are negative. The following lags seem to vary in a rather random fashion around zero.

The same type of diagram for the larger *SXBIG* set is shown in Figure 8.2. Surprisingly, the result for these 207 stocks shows a significantly *negative* correlation

in the first lag, whereas *positive* correlation is found for the 32 major stocks in *SXG*. We do not give any explanation for this observation, but it is reasonable to suspect a "small-company effect" in the larger *SXBIG* set. As a reference, the corresponding graphs for 100 simulated random-walk stocks are presented in Figure 8.3. The autocorrelation is very close to zero in a statistically stable way. This reinforces the results shown in the previous graphs for the real stock data.

Table 8.2: Mean k-step returns for 32 major Swedish stocks (SXG)

	k						
	1	**2**	**5**	**10**	**20**	**50**	**100**
Mean	0.091	0.183	0.445	0.892	1.831	4.585	9.142
Median	0.000	0.017	0.140	0.494	1.447	4.025	7.133
Std. dev	2.18	3.17	5.09	7.29	10.65	18.23	28.42
Skewness	0.52	0.70	0.78	0.71	0.56	0.53	0.64
Kurtosis	12.77	12.36	12.46	10.86	8.61	6.12	5.51
No of points	2178	2175	2172	2165	2153	2124	2074
Returns=0 (%)	14.1	9.3	5.5	3.6	2.4	1.2	0.8
Returns>0 (%)	43.6	46.1	49.5	52.1	56.1	60.2	62.6
Returns<0 (%)	42.2	44.6	45.0	44.3	41.5	38.5	36.6
Up fraction (%)	50.8	50.9	52.3	54.0	57.5	61.0	63.1
Mean Up	1.8	2.6	4.1	5.9	8.6	15.3	24.8
Mean Down	-1.6	-2.2	-3.5	-4.8	-7.0	-11.5	-16.0

Table 8.3: Mean k-step returns for 207 Swedish stocks (SXBIG)

	k						
	1	**2**	**5**	**10**	**20**	**50**	**100**
Mean	0.143	0.274	0.585	1.058	2.007	4.584	8.651
Median	0.000	0.007	0.060	0.248	0.946	2.895	5.148
Std. dev	3.02	4.15	6.15	8.42	11.80	18.82	27.80
Skewness	0.79	1.06	1.02	0.93	0.83	0.78	0.82
Kurtosis	15.78	16.49	11.55	9.27	7.58	5.99	5.59
No of points	1367	1363	1356	1347	1333	1306	1259
Returns=0 (%)	23.4	17.0	10.8	7.5	4.9	2.7	1.9
Returns>0 (%)	38.7	42.0	45.6	48.5	52.1	56.1	57.6
Returns<0 (%)	37.9	41.1	43.6	44.1	43.0	41.2	40.5
Up fraction (%)	50.6	50.6	51.1	52.3	54.7	57.6	58.7
Mean Up	2.7	3.5	5.2	7.1	10.1	16.8	26.5
Mean Down	-2.3	-2.9	-4.0	-5.3	-7.3	-11.3	-15.5

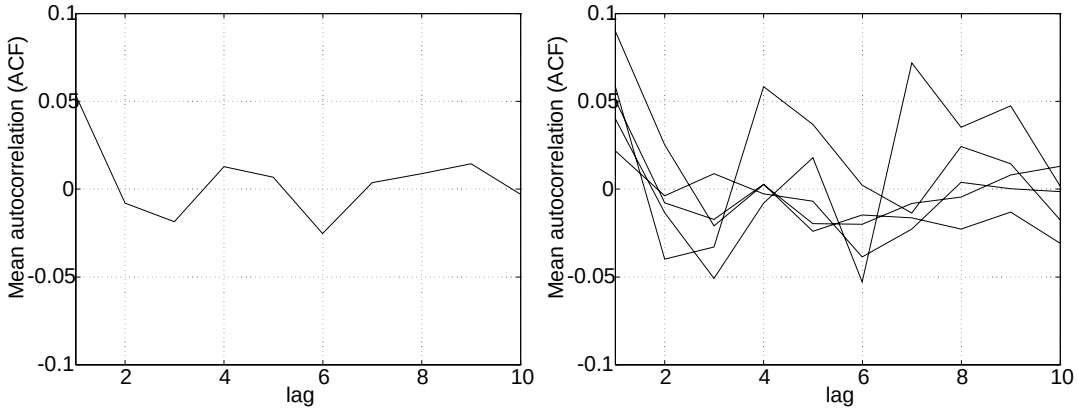

Figure 8.1: The autocorrelation for 32 major Swedish stocks (*SXG*) for the years 1987 to 1996. The graph on the right is broken down to five two-year curves

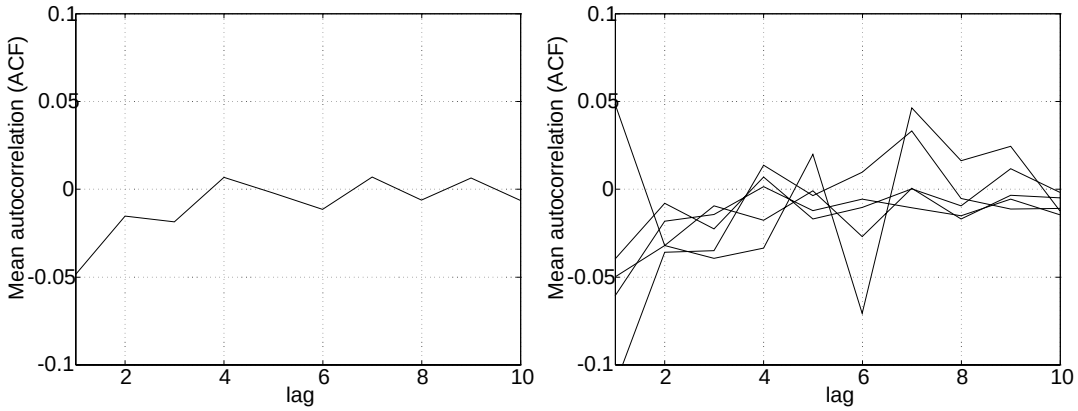

Figure 8.2: The autocorrelation for 207 Swedish stocks (*SXBIG*) for the years 1987 to 1996.

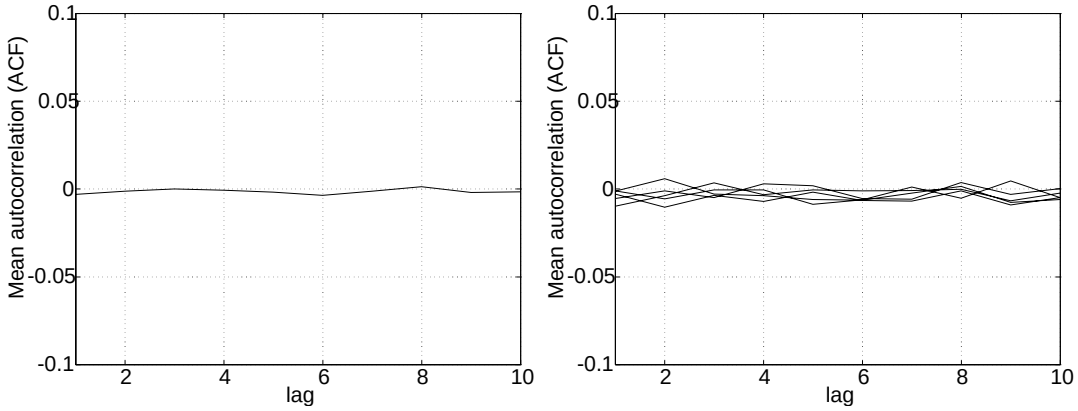

Figure 8.3: The autocorrelation for 100 simulated random-walk stocks for the years 1987 to 1996

Chapter 9

Trends

A trend-following trading strategy means buying stocks that have shown a positive trend for the last days, weeks or months. It also suggests selling stocks that have shown a negative trend. In this section the relevance for such a strategy is tested statistically.

A trend $T_k(t)$ was defined in (4.4) using the k-step return. By setting different values to k, we get measures indicating the growth of the stock per day, since its value k days ago.

To see if $T_k(t)$ is connected to future changes, define the profit $P_h(t)$ computed h days ahead as:

$$P_h(t) = 100 \cdot \frac{y(t+h) - y(t)}{y(t)}. \tag{9.1}$$

$P_h(t)$ is obviously equal to $R_h(t+h)$ (i.e. it is achieved by shifting the returns h days backwards).

In Table 9.1 the mean profit $P_1(t)$ is tabulated as a function of the trend $T_k(t)$, i.e. 1-step-forward profit versus k-step-backward trends. Results are presented for the *SXG* stocks for the years 1987-1996. Table 9.2 shows the "$Up\ fraction$" ((8.2)) and Table 9.3 the number of observations in each table entry. The label for each column is the mid-value of asymmetrical interval. For example, the column labeled **0.00** includes points with k-day trend in the interval $[-0.25\ 0.25[$. The intervals for the outermost columns are open ended on one side. Tables 9.4, 9.5, and 9.6 show the same, but with $P_5(t)$ and $T_k(t)$, i.e. 5-step-forward profit per day versus k-step-backward trends per day. The six Tables 9.7, 9.8, 9.9, 9.10, 9.11, and 9.12, repeat the same analysis for the larger *SXBIG* set.

To ensure that found patterns reflect fundamental properties of the process generating the data, and not only idiosyncrasies in the data, the relations between trends and future returns are also presented in graphs, in which one curve represents one year. Figure 9.1 shows 1-step profits versus 1-step trends for *SXG* and *SXBIG* respectively. Figure 9.2 shows 2-step profit versus k-step trends. Figure 9.3 shows 5-step profit versus k-step trends.

Let us draw some conclusions from these statistical examinations of trends.

- The massive better part of returns falls into a region, where it is very difficult to claim any correlation between past and future price changes. The regions, where any correlation may be significant, are the sparsely populated extreme ones. However, some interesting effects can be observed. Looking at Table 9.8, we observe that a 5% decrease in price since the previous day, stands a 61.2% probability of showing an increase by the following day. The same negative correlation holds for both sets of investigated stocks.

- The cases with large increase since the previous day call for a more complex interpretation. Looking at Figure 9.2, a difference between the two investigated sets of stocks becomes apparent. For the stocks in *SXG*, a 5% increase in price since the previous day stands a 54.9% probability of showing an *increase* by the next day. For the stocks in *SXBIG*, a 5% increase in price since the previous day stands a 53.3% probability of showing a *decrease* by the following day. This is in accordance with the observed difference in the first lag of the ACF, reported in Section 8.2.

Table 9.1: Mean 1-step returns for 32 stocks in SXG

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	0.52	-0.03	-0.05	-0.05	-0.12	-0.08	0.00	0.11	0.19	0.28	0.37	0.40	0.73
2	1.19	0.20	0.13	-0.09	-0.06	-0.08	0.03	0.13	0.24	0.31	0.28	0.31	0.43
3	1.87	0.06	0.15	0.16	-0.04	-0.07	0.03	0.15	0.20	0.25	0.27	-0.02	0.38
4	3.17	0.48	-0.03	0.12	-0.00	-0.04	0.03	0.16	0.20	0.16	0.42	-0.04	0.83
5	3.47	0.93	0.19	-0.03	0.02	-0.01	0.05	0.15	0.20	0.14	0.21	0.57	1.12
10	10.64	7.39	0.38	-0.11	0.00	-0.00	0.06	0.15	0.17	0.30	0.29	0.44	0.56
20			8.25	1.23	-0.14	-0.06	0.09	0.14	0.21	0.31	0.06	3.28	1.29
30				2.63	0.02	-0.07	0.09	0.13	0.22	0.33	2.19	1.24	1.49
50					0.48	-0.08	0.08	0.13	0.11	1.10	-0.17	1.95	1.21
100					4.68	0.18	0.05	0.13	0.26	0.57	0.78	0.29	0.81

Table 9.2: Fraction up/(up+down) moves (%) for stocks in SXG

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	55.9	50.6	49.6	48.5	47.1	47.3	49.1	51.4	53.3	54.9	55.6	55.0	54.9
2	58.3	51.9	53.1	49.5	47.6	47.8	49.5	51.8	54.5	54.4	52.9	51.8	48.8
3	62.2	51.4	49.5	53.3	48.7	47.3	49.8	52.6	53.6	53.6	51.5	42.4	47.9
4	64.8	55.7	48.9	50.4	49.9	47.9	50.0	52.8	54.1	50.8	49.4	43.0	50.6
5	60.6	58.9	51.0	49.8	50.0	48.9	50.1	52.8	52.7	50.0	48.3	49.1	53.8
10	100.0	73.3	52.6	50.3	49.3	48.8	50.8	52.8	50.4	51.8	50.2	47.5	43.5
20			77.8	56.1	47.1	47.5	51.4	52.3	50.7	48.6	49.1	50.0	55.6
30				56.1	47.6	47.3	51.5	51.7	50.2	50.7	55.8	61.1	54.8
50					52.3	46.3	51.2	51.8	48.8	54.4	52.8	57.1	50.0
100					71.4	49.5	50.5	51.6	49.2	54.8	64.7	50.0	53.5

Table 9.3: Number of points (SXG)

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	1396	1178	2539	6427	8701	7543	12373	7317	8508	6443	3048	1504	1807
2	492	561	1553	4644	9334	10822	12800	10219	9335	5399	2008	759	796
3	244	346	985	3514	8807	11837	15369	11419	9238	4575	1415	478	456
4	116	215	716	2717	8246	12485	17295	12122	9272	3865	950	317	310
5	71	136	553	2178	7478	13224	18504	12871	9185	3215	702	244	218
10	3	30	165	957	4794	13670	23902	15643	7177	1536	292	93	79
20	0	0	9	359	2548	12327	29970	17393	4441	742	123	18	27
30	0	0	0	70	1855	10829	33884	17278	3156	473	48	18	32
50	0	0	0	0	765	8802	39328	15867	1971	216	41	24	42
100	0	0	0	0	7	5798	46293	12148	861	258	40	5	48

Table 9.4: Mean 5-step returns for 32 stocks in SXG

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	1.14	0.05	0.44	0.18	0.21	0.14	0.31	0.43	0.65	0.67	0.64	0.74	1.22
2	1.87	0.80	0.67	0.29	0.23	0.23	0.26	0.50	0.62	0.68	0.44	0.61	1.36
3	2.66	0.68	0.88	0.62	0.30	0.15	0.29	0.52	0.60	0.65	0.60	0.30	1.57
4	4.28	0.89	1.45	0.52	0.27	0.17	0.35	0.52	0.66	0.50	1.15	-0.41	2.55
5	6.30	1.44	1.09	0.30	0.28	0.23	0.35	0.56	0.62	0.56	0.52	1.69	2.17
10	9.84	5.09	1.27	0.01	0.21	0.16	0.41	0.58	0.74	0.72	1.03	0.51	2.69
20			16.02	3.05	-0.27	-0.09	0.44	0.66	0.78	1.27	-1.05	2.80	8.47
30				6.04	-0.13	-0.21	0.47	0.60	0.86	1.67	6.70	-1.06	6.76
50					1.84	-0.26	0.39	0.59	0.76	3.10	4.13	9.78	4.26
100					7.96	1.01	0.25	0.51	1.32	3.25	1.91	4.36	3.81

Table 9.5: Fraction up/(up+down) moves (%) for stocks in SXG

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	56.7	51.3	51.2	50.7	50.4	50.0	52.0	52.7	54.6	54.1	53.4	52.5	51.4
2	55.9	55.4	55.3	51.7	50.3	51.1	51.5	53.0	54.0	54.1	50.6	50.2	49.3
3	56.4	54.5	55.6	55.0	51.5	50.1	51.9	53.2	53.7	53.2	49.8	48.7	46.1
4	62.6	55.3	59.1	54.8	51.8	50.5	52.1	52.8	54.2	50.7	51.6	43.4	50.2
5	65.2	55.5	57.1	53.3	52.0	51.0	52.0	53.4	53.5	50.7	48.0	52.2	47.5
10	66.7	62.1	50.3	49.3	51.4	49.7	52.9	53.8	52.7	51.1	50.4	51.2	41.3
20			100.0	59.2	46.6	48.1	53.4	54.3	50.3	50.5	40.2	50.0	56.0
30				69.1	46.1	47.4	53.5	53.0	51.7	52.2	63.0	41.2	62.1
50					55.7	45.7	52.8	53.4	49.1	60.0	63.4	63.6	64.1
100					83.3	51.0	51.7	52.9	54.5	64.8	61.1	80.0	63.6

Table 9.6: Number of points (SXG)

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	1382	1173	2549	6406	8682	7519	12367	7288	8467	6415	3022	1494	1793
2	486	558	1536	4638	9315	10783	12763	10172	9300	5369	2001	753	793
3	237	340	966	3507	8776	11792	15363	11392	9204	4541	1400	469	458
4	113	209	709	2710	8216	12459	17253	12102	9216	3832	934	315	312
5	71	133	547	2177	7442	13193	18487	12846	9143	3186	701	244	211
10	3	30	164	946	4774	13595	23856	15642	7154	1529	289	90	80
20	0	0	8	361	2533	12329	29881	17312	4434	735	123	17	27
30	0	0	0	70	1850	10819	33799	17193	3145	467	48	17	32
50	0	0	0	0	760	8777	39226	15803	1960	214	43	24	42
100	0	0	0	0	7	5771	46183	12106	861	259	40	5	48

Table 9.7: Mean 1-step returns for 207 stocks in SXBIG

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	1.42	0.40	0.30	0.09	-0.02	-0.07	0.02	0.09	0.11	0.14	0.15	0.16	0.06
2	2.35	0.75	0.44	0.22	0.06	-0.00	0.02	0.06	0.09	0.13	0.06	0.10	0.05
3	3.38	0.95	0.62	0.35	0.12	0.01	0.02	0.07	0.11	0.11	-0.00	-0.12	-0.12
4	4.47	1.36	0.65	0.45	0.14	0.04	0.02	0.07	0.12	0.04	-0.00	0.07	-0.03
5	5.06	2.21	0.87	0.40	0.20	0.04	0.04	0.10	0.10	0.05	-0.07	-0.03	-0.01
10	8.28	5.02	1.88	0.57	0.24	0.08	0.06	0.10	0.11	0.13	0.18	0.58	-0.89
20	40.22	11.58	2.43	1.84	0.26	0.05	0.08	0.11	0.16	0.15	-0.10	-0.02	-1.07
30			8.07	3.41	0.38	0.09	0.08	0.10	0.19	0.21	-0.12	0.59	-1.37
50				7.55	0.81	0.08	0.10	0.14	0.11	0.01	0.37	-0.71	-1.05
100					2.51	0.22	0.08	0.15	0.31	-0.15	-0.53	-2.10	-0.30

Table 9.8: Fraction up/(up+down) moves (%) for stocks in SXBIG

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	61.2	56.9	55.0	52.1	49.6	48.0	48.9	50.2	50.0	49.8	49.8	50.0	46.7
2	62.8	57.6	56.9	53.7	51.1	49.5	49.2	49.3	49.8	50.1	48.3	47.5	45.7
3	65.0	58.4	56.5	55.7	51.9	49.4	49.2	49.7	50.1	49.4	47.3	44.6	43.8
4	64.6	58.9	55.5	55.2	52.5	50.1	49.0	49.9	51.0	48.3	46.1	45.8	44.7
5	65.4	60.8	55.7	54.5	52.8	50.0	49.5	50.5	50.2	47.8	44.9	45.5	44.1
10	65.0	65.1	56.0	53.0	51.9	50.6	50.3	50.7	48.9	47.8	46.2	47.9	38.4
20	66.7	65.4	54.8	56.9	50.2	49.5	50.8	50.7	49.1	47.8	43.7	44.0	40.6
30			55.2	57.5	49.4	49.3	51.0	50.4	49.3	47.3	39.8	53.6	37.7
50				57.7	50.5	48.4	50.7	51.0	48.2	46.9	47.9	37.5	41.7
100					52.4	49.1	50.3	50.7	49.3	46.9	47.7	31.6	50.0

Table 9.9: Number of points (SXBIG)

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	10256	6255	12158	23991	29276	19134	60845	18447	28262	23213	13091	7077	13141
2	4071	3206	7475	19783	34386	35228	53625	33001	32519	20794	9214	4335	6703
3	2121	2101	5000	15661	33500	42333	58688	39244	32600	18332	6854	2994	4244
4	1187	1426	3640	12431	31945	46192	64660	42539	32581	15915	5432	2264	2982
5	744	990	2852	10142	29791	49017	69243	45101	32426	13989	4384	1698	2344
10	116	255	1047	4837	19894	52570	89411	53333	27421	8062	2238	807	890
20	3	28	168	1941	11475	47613	112230	59258	19290	4344	1029	338	320
30	0	0	32	708	8339	42626	127005	58767	14641	3021	625	209	179
50	0	0	0	59	4033	34769	147072	54372	10147	1655	311	115	83
100	0	0	0	0	364	23997	167490	44255	5972	1094	129	20	53

Table 9.10: Mean 5-step returns for 207 stocks in SXBIG

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	2.30	0.82	0.66	0.38	0.27	0.18	0.41	0.47	0.53	0.56	0.54	0.63	0.21
2	4.32	1.56	0.86	0.57	0.32	0.26	0.40	0.49	0.55	0.48	0.27	0.56	-0.16
3	5.89	1.67	1.36	0.70	0.38	0.24	0.41	0.51	0.54	0.50	0.26	0.61	-0.22
4	7.51	3.37	1.42	0.82	0.37	0.26	0.41	0.51	0.60	0.41	0.47	0.36	-0.16
5	9.31	3.75	1.95	0.73	0.39	0.28	0.40	0.60	0.55	0.46	0.30	0.63	-0.56
10	16.56	12.35	4.29	0.86	0.33	0.22	0.47	0.60	0.65	0.44	0.84	0.47	-3.16
20	78.89	11.97	13.65	4.96	0.18	0.06	0.44	0.63	0.76	1.36	0.20	-0.42	-4.85
30			23.96	8.33	0.64	0.08	0.45	0.59	0.95	1.27	1.61	-2.39	-6.15
50				12.18	2.39	0.01	0.41	0.73	0.71	0.92	0.73	-2.77	-5.85
100					8.33	0.68	0.33	0.68	1.08	0.58	-1.84	-4.26	2.10

Table 9.11: Fraction up/(up+down) moves (%) for stocks in SXBIG

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	58.6	54.4	53.8	51.3	50.4	49.6	50.2	52.3	51.4	50.7	49.4	48.7	44.1
2	61.6	56.3	55.9	53.4	50.7	50.1	50.3	51.3	51.2	49.7	46.5	46.8	41.5
3	63.5	57.1	56.9	54.9	51.6	49.9	50.6	51.2	50.8	48.6	45.6	44.4	41.3
4	63.5	60.7	56.4	55.5	51.9	50.2	50.6	51.1	51.2	47.4	45.1	43.1	40.7
5	64.7	62.5	56.5	54.7	52.4	50.1	50.7	51.7	50.4	46.9	44.5	44.1	39.2
10	74.0	67.5	57.4	52.1	51.3	49.8	51.4	51.8	49.6	45.9	44.5	43.7	34.6
20	100.0	53.6	66.9	58.6	48.3	48.5	51.6	51.9	49.2	48.9	40.4	38.4	31.6
30			58.1	60.2	48.0	48.3	51.8	51.0	49.4	47.7	41.6	35.7	30.3
50				47.4	51.4	46.8	51.2	52.3	47.3	45.9	43.6	38.5	38.7
100					58.2	48.8	50.4	51.9	49.7	45.3	44.5	36.8	58.3

Table 9.12: Number of points (SXBIG)

	k-day trend (%/day)												
k	**-5.00**	**-4.00**	**-3.00**	**-2.00**	**-1.00**	**-0.50**	**0.00**	**0.50**	**1.00**	**2.00**	**3.00**	**4.00**	**5.00**
1	10120	6203	12085	23788	29099	19057	60477	18284	27958	22945	12888	6969	12834
2	4000	3177	7442	19726	34260	35003	53355	32659	32188	20508	9050	4255	6453
3	2029	2081	4916	15631	33442	42073	58477	38851	32278	17965	6713	2932	4049
4	1137	1391	3576	12334	31909	45973	64282	42165	32168	15663	5278	2197	2862
5	713	965	2790	10055	29701	48998	68890	44674	31987	13759	4290	1648	2234
10	111	256	1010	4736	19775	52317	88853	53000	27101	7880	2200	782	862
20	2	28	167	1919	11375	47582	111654	58661	19034	4241	999	327	305
30	0	0	31	693	8232	42555	126516	58131	14367	2957	602	207	175
50	0	0	0	59	3971	34558	146319	53863	9990	1623	311	111	83
100	0	0	0	0	359	23859	166617	43693	5919	1079	121	20	52

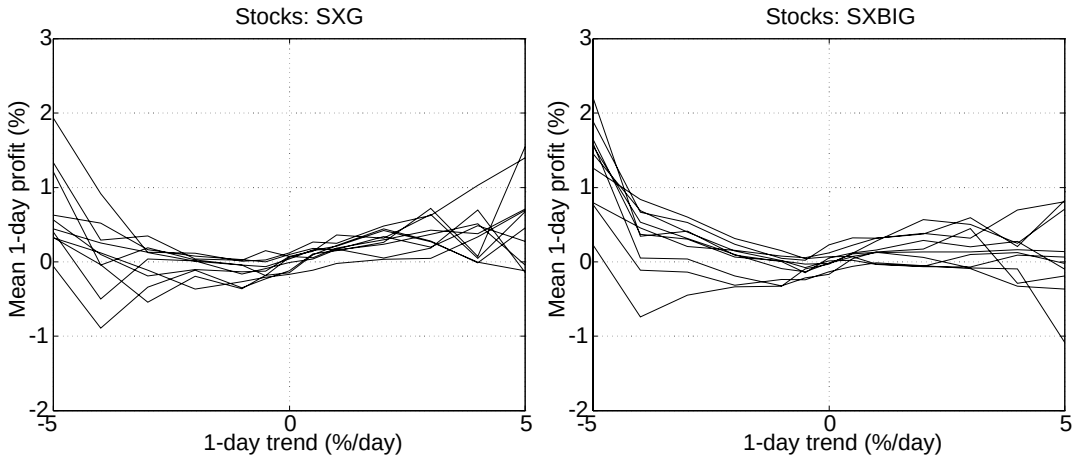

Figure 9.1: 1-step profits versus 1-step returns for stocks *SXG* and *SXBIG*. Each curve represents one year between 1987 and 1996.

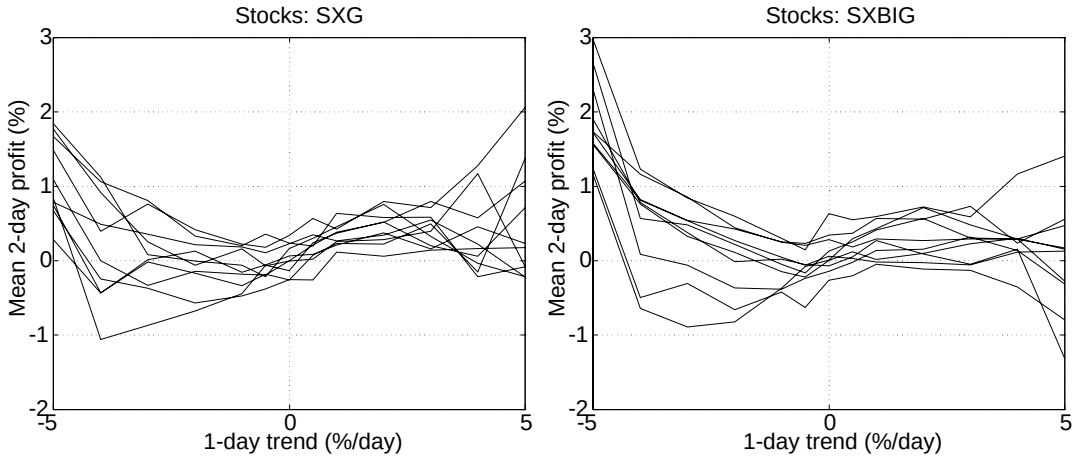

Figure 9.2: 2-step profits versus 1-step returns for stocks *SXG* and *SXBIG*. Each curve represents one year between 1987 and 1996.

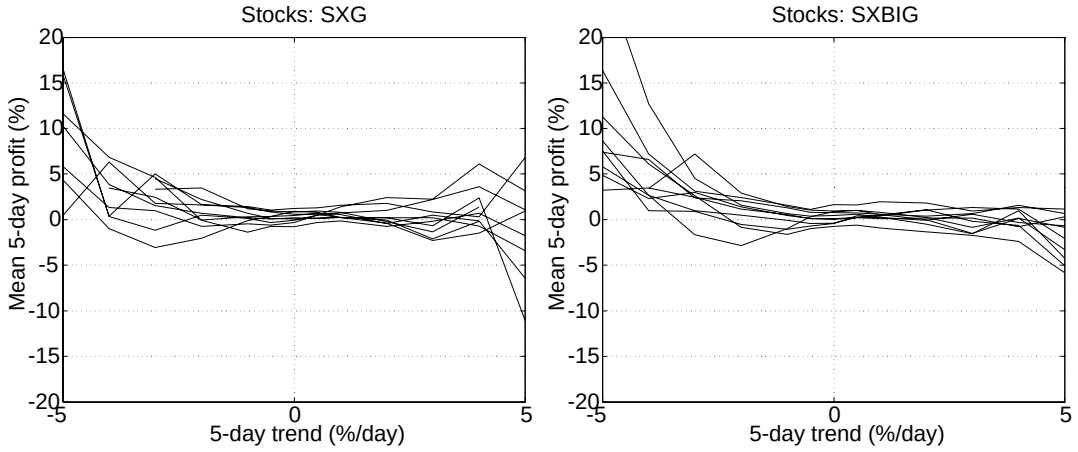

Figure 9.3: 5-step profits versus 5-step returns for the *SXG* and *SXBIG* stocks. Each curve represents one year between 1987 and 1996.

Chapter 10

Seasonal Effects

In this section we present some empirical investigations showing how the day of the week and the month of the year affect the stock price returns. The results confirm and complement similar investigations on other stock markets world-wide.

10.1 Handling the Missing Weekends

The exchanges in most countries are closed on Saturday and Sunday. The price time series therefore have gaps for these days if plotted against a calendar-time axis. The situation is normally approached in either of the two following ways:

- The returns from Friday to Monday are assumed to be three times the returns on each of the other days of the week. This is called the calendar-time hypothesis
- Returns are generated only during active trading days, and average returns are the same for all five days of the week. This is called the trading-time hypothesis.

Hawawini and Keim [16] report on negative Monday returns (i.e. relative price changes from Friday close to Monday close) for several European stock markets. This is inconsistent with both the calendar-time hypothesis and trading-time hypothesis. A satisfactory explanation for this weekend effect does not exist, according to Hawawini and Keim, at present.

Our own results do not support any of the mentioned hypotheses either. However, the day-to-day returns exhibit a complex pattern, as shown in the next section. To proceed we have to define several metrics for a stock price time series $\{y(0), y(1), y(2), ..., y(N)\}$.

10.1.1 One-Step and One-Day Returns

The **one-step** returns $R_S(t)$ are defined as the relative increase in price since the prior point in the time series:

$$R_S(t) = 100 \cdot \frac{y(t) - y(t-1)}{y(t-1)}. \tag{10.1}$$

The mean of one-step returns $R_S(t)$ is computed using the standard definition as:

$$R_S = \frac{1}{N}\sum_{t=1}^{N} R_S(t). \tag{10.2}$$

To deal with the holiday effect we propose the concept of **one-day** returns $R_D(t)$. It is defined as the relative increase in price since the previous day, if that day is represented in the time series:

$$R_D(t) = 100 \cdot \delta(t) \cdot \frac{y(t) - y(t-1)}{y(t-1)}, \tag{10.3}$$

where:

$$\delta(t) = \begin{cases} 1 \text{ if } juliandate(t) - juliandate(t-1) = 1 \\ 0 \text{ otherwise} \end{cases}. \tag{10.4}$$

The function *juliandate(t)* returns the number of days since the start of the Julian calendar up to the date for sample t. It is introduced as a means to compute the number of days between two samples.

The mean of the **one-day** returns $R_D(t)$ is:

$$R_D = \frac{1}{K}\sum_{t=1}^{N} R_D(t), \tag{10.5}$$

where:

$$K = \sum_{t=1}^{N} \delta(t). \tag{10.6}$$

K is roughly the number of weekdays Tuesday-Friday in the data set. Holidays occurring on weekdays as well as missing data also generate $\delta = 0$, and therefore a reduced value on K.

The standard deviations for one-day returns and one-step returns are computed correspondingly.

10.1.2 Increase Fraction

We also define increase and decrease fractions based on the standard one-step returns. The **one-step** Increase Fraction I_S is defined as:

$$I_S = \frac{1}{K_S}\sum_{t=1}^{N} a_I(t), \tag{10.7}$$

where

$$a_I(t) = \begin{cases} 1 \text{ if } y(t) > y(t-1) \\ 0 \text{ otherwise} \end{cases} \tag{10.8}$$

and

$$K_S = \sum_{t=1}^{N} a_I(t) + a_D(t). \tag{10.9}$$

The **one-step** Decrease fraction D_S is defined as:

$$D_S = \frac{1}{K_S}\sum_{t=1}^{N} a_D(t), \tag{10.10}$$

where

$$a_D(t) = \begin{cases} 1 \text{ if } y(t) < y(t-1) \\ 0 \text{ otherwise} \end{cases} . \tag{10.11}$$

To deal with the holiday effect, we define in a similar fashion increase and decrease fractions based on the one-day returns. The **one-day** Increase Fraction I_D is defined as:

$$I_D = \frac{1}{K_D} \sum_{t=1}^{N} \delta(t) \cdot a_I(t), \tag{10.12}$$

where

$$K_D = \sum_{t=1}^{N} \delta(t) \cdot (a_I(t) + a_D(t)). \tag{10.13}$$

The **one-day** Decrease fraction D_D is defined as:

$$D_D = \frac{1}{K_D} \sum_{t=1}^{N} \delta(t) \cdot a_D(t). \tag{10.14}$$

Some simple statistics for the one-day and one-step returns are presented below. The results in Table 10.1 are mean values for the *SXG* stocks and the results in Table 10.2 are mean values for the *SXBIG* stocks. Results are presented for the years 1987-1996.

Table 10.1: Average daily returns for 32 Swedish stocks (*SXG*)

One-Day-Returns $R_D(t)$			One-Step-Returns $R_S(t)$		
Mean	Std.dev.	No. of obs. [3]	Mean	Std.dev.	No. of obs. [3]
0.103	2.20	54696	0.084	2.25	69687

Table 10.2: Average daily returns for 207 Swedish stocks (*SXBIG*)

One-Day-Returns $R_D(t)$			One-Step-Returns $R_S(t)$		
Mean	Std.dev.	No. of obs. [3]	Mean	Std.dev.	No. of obs. [3]
0.132	3.25	222727	0.123	3.31	283008

The results show differences in both mean returns and variances for the two types of returns computed. The difference in the mean values is caused by a weekend effect and investigated in more detail in the next section. The difference in variances shows that the one-day returns exhibit a more homogenous distribution than the one-step

returns. It can be interpreted as a reduction in noise in the computation of $R_D(t)$ by removing the weekends, holidays, and other missing data.

Rejection of the trading-time hypothesis should call for some sort of special treatment when creating models for the prediction of future returns. However, this is seldom seen in published methods. Some researchers have noticed the problem, which seems to be even more prominent in the case of prediction of exchange rates. Weigend [39] deals with the situation by simply removing all but the Monday-to-Tuesday returns and performs the prediction task on this aggregated time series. Another way to deal with the problem would be to include the day of the week explicitly in the model.

10.2 Day-of-the-Week Effect

Friday night arrives without a suitcase
Lady Madonna, Lennon/McCartney

The day-of-the-week effect has been studied in a number of research papers. Hawawini and Keim [16] present a summary, which demonstrates significant differences in average daily returns across days of the week. In our investigation of the Swedish stock market, the daily returns are presented in a somewhat different fashion than is normally done. The stock returns are computed for all twenty-five combinations of buy and sell days. The returns are presented as "daily returns", i.e. they are divided by the number of calendar days between buy and sell. For example, the return from buying on Friday and selling on Monday is divided by three before it is put in the table. A second table with the same layout presents the "Increase Fraction" for the same combinations of buy and sell days. This number is the fraction of sampled buy/sell situations that resulted in a positive return (zero returns are removed before calculating the Increase Fraction). In this way, all combinations of buy and sell days can be compared on an equal basis.

As before, results are presented for the *SXG* set of stocks. A time period of 10 years (1987-1996) was used to generate the results. This provides statistically more stable grounds than using one single index as in the investigations presented in [16]. The daily returns R_D are shown in Table 10.3 and the Increase Fraction I_D in Table 10.4.

The same type of tables as the above have been generated for the *SXBIG* stocks as well. The daily returns R_D are shown in Table 10.5 and the Increase Fraction I_D in Table 10.6.

We can extract several interesting "anomalies" from the last two tables:

- The day-of-the-week affects the returns significantly. The returns span between 0.003% (buy Friday/sell Tuesday) and 0.243% (buy Thursday/sell Friday).

- The one-day returns increase monotonically from Monday to Thursday: 0.004, 0.114, 0.167, 0.243 (buying on a Friday never yields a one-day return).

[3] The ratio between the two No.-of- obs columns should be 4/5 if only Friday-to-Monday returns have been removed. However, there are additional gaps in the data series originating from holidays and simply missing data.

Table 10.3: Daily returns (%) for combinations of Buy and Sell days for SXG

	Sell Day					
Buy Day	Mon	Tue	Wed	Thu	Fri	**Mean**
Mon	0.045	0.005	0.033	0.078	0.088	0.050
Tue	0.062	0.056	0.094	0.138	0.133	0.097
Wed	0.054	0.048	0.057	0.174	0.159	0.099
Thu	0.023	0.018	0.033	0.049	0.138	0.052
Fri	-0.013	-0.012	0.002	0.027	0.046	0.010
Mean	0.034	0.023	0.044	0.093	0.113	0.061

Table 10.4: Increase fraction (%) for combinations of Buy and Sell days for SXG

	Sell Day					
Buy Day	Mon	Tue	Wed	Thu	Fri	**Mean**
Mon	50.58	48.99	49.88	51.20	51.67	50.46
Tue	51.13	52.05	50.73	52.42	52.61	51.79
Wed	51.38	51.75	52.83	53.08	52.86	52.38
Thu	49.20	49.72	50.87	51.61	51.27	50.53
Fri	48.41	48.02	48.99	50.24	51.43	49.42
Mean	50.14	50.11	50.66	51.71	51.97	50.92

Table 10.5: Daily returns (%) for combinations of Buy and Sell days for SXBIG

	Sell Day					
Buy Day	Mon	Tue	Wed	Thu	Fri	**Mean**
Mon	0.069	0.004	0.050	0.078	0.105	0.061
Tue	0.081	0.066	0.114	0.136	0.152	0.110
Wed	0.076	0.060	0.066	0.167	0.186	0.111
Thu	0.061	0.044	0.051	0.058	0.243	0.091
Fri	0.014	0.003	0.018	0.033	0.062	0.026
Mean	0.060	0.035	0.060	0.095	0.150	0.080

Table 10.6: Increase fraction (%) for combinations of Buy and Sell days for SXBIG

	Sell Day					
Buy Day	Mon	Tue	Wed	Thu	Fri	**Mean**
Mon	50.21	48.42	49.07	49.89	50.66	49.65
Tue	50.85	50.68	50.39	51.24	52.04	51.04
Wed	50.79	50.27	51.16	51.63	52.40	51.25
Thu	49.80	49.38	49.86	50.41	52.05	50.30
Fri	48.70	47.51	48.18	49.02	50.30	48.74
Mean	50.07	49.25	49.73	50.44	51.49	50.20

- The rightmost column describes the mean returns achieved when selling between one and seven days from the buying day. Friday appears to be the worst day to buy in this respect.

- Looking at "Increase Fractions," it is still clear that the real trading conditions are almost as bad as before. Even if we pick the best choice and buy on Thursday and sell on Friday, we loose money in 47.95% of the cases. It would take great patience and a stable financial backup to utilize the shown day-of-the-week effect.

A question that should be posed always when looking for and finding structures in huge data sets, is whether the found structure reflects some general property of the data generating process, or is simply an effect of data snooping. The general question is hard to answer, at the least, and is further discussed in Chapter 3. In this particular case, we have calculated the same statistics for yearly data over 1987-1996. In this way, the results are tested for stability in time. The reported effects are present even in these cases, and thus provide additional support for the results.

10.3 Month Effects

The month effect on stock returns is investigated in a similar fashion as the day-of-the-week effect above. We perform two statistical investigations:

- Average daily returns for each month

- Average monthly returns for combinations of buy and sell months

10.4 Daily Returns for Each Month

In this study we compute average daily returns $R_D(t)$ and $R_S(t)$ as defined in Section 10.1. The reason against using the "normal" $R_S(t)$ is the day-of-the-week effect, shown in the previous two sections. By using the weekend compensated return $R_D(t)$, we achieve more stable figures with the "noise" caused by weekends, holidays, and missing data removed. The returns are computed for the *SXG* and *SXBIG* stocks

for the years 1987-1996. The mean results for *SXG* are shown in Tables 10.7 and 10.8. The results for *SXBIG* are shown in Tables 10.9 and 10.10.

The results for the Swedish stock market fits well with investigations on other markets. Hawawini and Keim [16] present a summary of a research on a number of stock markets world-wide. The high returns for January and low returns for September are significant for most of the markets, including the Swedish stock market.

10.5 Increase Fractions

The Increase Fraction I_D for *SXBIG* varies between 47.80% (March) and 54.44% (January). The mean I_D is 50.62%, which is close to the 50%, proposed by the random-walk hypothesis. Note, that a prediction accuracy ("hit rate") of about 54% is often reported for elaborate prediction algorithms. Most algorithms do not use the day-of-the-week or the month-of-the-year as input variables, see e.g. [37] or [4]. These results are claimed to show predictive capability in the algorithms. Be that as it may, if we can achieve the same hit rate by just looking at what month we are trading in, it seems reasonable to incorporate in some way the month-of-the-year in the algorithm. And the validation process really should be reconsidered for algorithms that do not do that.

Table 10.7: One-day returns for SXG

	Mean	**Std.dev.**	**Incr.fraction**	**No. of obs.**
Jan	0.355	2.37	55.92	4338
Feb	0.173	2.24	52.19	4310
Mar	0.065	1.95	48.62	4748
Apr	0.227	1.85	54.78	4130
May	0.133	2.06	52.22	4078
Jun	0.007	1.75	47.71	4426
Jul	0.194	1.61	55.86	4887
Aug	-0.002	2.17	48.17	5059
Sep	-0.028	2.37	49.72	4750
Oct	-0.019	2.73	48.33	5015
Nov	0.106	2.70	49.58	4872
Dec	0.076	2.18	50.52	4083
Mean	0.103	2.20	51.03	4558

Table 10.8: One-step returns for SXG

	Mean	**Std.dev.**	**Incr.fraction**	**No. of obs.**
Jan	0.331	2.48	55.07	5700
Feb	0.124	2.29	51.49	5499
Mar	0.034	1.98	48.31	6133
Apr	0.200	1.94	53.68	5486
May	0.178	2.08	53.17	5587
Jun	0.007	1.85	48.11	5709
Jul	0.176	1.71	54.68	6234
Aug	-0.105	2.27	46.29	6352
Sep	-0.038	2.49	48.65	6072
Oct	-0.102	2.88	48.22	6303
Nov	0.092	2.79	49.63	6170
Dec	0.081	2.32	50.99	5575
Mean	0.077	2.29	50.60	5902

Table 10.9: One-day returns for SXBIG

	Mean	**Std.dev.**	**Incr.fraction**	**No. of obs.**
Jan	0.376	3.32	54.44	17096
Feb	0.231	3.54	51.44	17307
Mar	0.019	2.93	47.80	19154
Apr	0.241	2.82	53.87	16224
May	0.143	3.02	51.28	16501
Jun	0.034	2.99	48.80	17750
Jul	0.237	2.64	54.10	18417
Aug	0.019	3.10	48.92	20524
Sep	0.041	3.38	50.03	19877
Oct	0.078	3.72	48.78	21181
Nov	0.133	3.52	49.58	20397
Dec	0.095	3.64	49.97	18303
Mean	0.132	3.25	50.62	18561

Table 10.10: One-step returns for SXBIG

	Mean	**Std.dev.**	**Incr.fraction**	**No. of obs.**
Jan	0.357	3.90	53.20	24058
Feb	0.208	4.03	50.76	23590
Mar	-0.025	3.43	47.35	26661
Apr	0.202	3.28	52.67	23408
May	0.152	3.47	51.62	24406
Jun	-0.026	3.23	48.10	24914
Jul	0.220	3.09	52.99	26042
Aug	-0.096	3.45	47.24	27941
Sep	-0.042	3.78	48.49	27338
Oct	0.036	4.57	48.77	28515
Nov	0.082	4.04	48.91	27764
Dec	0.111	4.56	50.33	26447
Mean	0.092	3.78	49.94	25924

10.6 Monthly Returns for Combinations of Buy and Sell Months

We conclude the investigations of seasonal effects with a trading-oriented statistical test, where both buying and selling are considered. The first trading day in each month is always selected for both buying and selling. After buying in the beginning of a month, the returns from selling in the beginning of each of the successive twelve months are stored in a twelve-by-twelve table. The shown figures in this table are daily returns times 30, to obtain comparable monthly returns for all months, regardless of the number of days they contain. A second table with the same layout shows the Increase Fraction for the same combinations of buy and sell months. This number is the fraction of sampled buy/sell situations that resulted in a positive return (zero returns are removed before calculating the Increase Fraction). In this way, all combinations of buy and sell months can be compared on an equal basis.

Tables 10.11 and 10.12 show the results for the *SXBIG* stocks for the years 1987-1996. Tables 10.13 and 10.14 show the results for the *SXG* stocks for the same time period. To avoid false conclusions caused by extreme trends during parts of single years, the data is also investigated in finer detail. Figures 10.2, 10.3, 10.4, 10.5, 10.6, and 10.7, present results for the 32 *SXG* stocks for the years 1987-1996. Each graph contains data for one specific buy month. Each point in the graphs shows the mean profit from selling all the stocks at that particular month in one particular year. Each point thus represents 32 trades. There are 10 points plotted for each sell month, and one for each year between 1987 and 1996. Both buying and selling are done once per month on the first trading day of the month. The detailed analysis for the *SXBIG* set can be found in [18]. The same combinations of buy and sell months are shown in the 3-D diagrams in Figure 10.1.

Table 10.11: Monthly returns (%) for combinations of buy and sell months for SXBIG

	Sell Month												
Buy Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	**Mean**
Jan	1.47	6.36	4.79	2.70	2.91	2.92	2.35	2.70	2.18	1.91	1.87	1.76	2.83
Feb	1.14	1.91	3.91	1.34	1.95	2.27	1.76	2.40	1.82	1.65	1.61	1.49	1.94
Mar	0.52	1.27	1.33	-0.58	1.12	1.60	1.10	1.77	1.22	1.01	1.00	0.94	1.03
Apr	0.73	1.52	1.51	1.12	3.31	2.80	1.70	2.46	1.59	1.27	1.23	1.13	1.70
May	0.35	1.25	1.27	0.87	1.18	2.56	0.90	2.07	1.09	0.78	0.82	0.73	1.16
Jun	-0.06	0.89	0.99	0.64	0.97	1.08	-0.65	1.49	0.51	0.27	0.26	0.29	0.56
Jul	0.16	1.20	1.32	0.85	1.19	1.28	1.13	4.01	1.11	0.62	0.57	0.54	1.16
Aug	-0.82	0.43	0.63	0.28	0.66	0.85	0.72	1.01	-1.53	-1.07	-0.68	-0.32	0.01
Sep	-0.49	1.06	1.24	0.76	1.13	1.31	1.15	1.54	1.28	-1.04	-0.45	0.08	0.63
Oct	-0.05	1.82	2.00	1.29	1.61	1.81	1.57	2.06	1.67	1.45	0.15	0.74	1.34
Nov	0.07	2.32	2.43	1.58	1.86	2.10	1.80	2.26	1.85	1.60	1.82	1.25	1.75
Dec	0.35	3.68	3.38	2.10	2.34	2.51	2.09	2.48	1.97	1.70	1.90	1.85	2.20
Mean	0.28	1.98	2.07	1.08	1.69	1.92	1.30	2.19	1.23	0.85	0.84	0.87	1.36

Table 10.12: Increase fraction (%) for combinations of buy and sell months for SXBIG

	Sell Month												
Buy Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	**Mean**
Jan	53.89	65.12	66.08	59.48	66.08	66.28	64.06	66.46	61.84	62.26	59.27	59.68	62.54
Feb	50.37	52.39	55.03	53.02	58.68	59.53	59.04	61.10	57.50	57.52	55.35	54.84	56.20
Mar	46.57	48.89	50.81	43.03	54.95	58.05	56.05	58.75	54.41	55.24	51.90	50.84	52.46
Apr	49.61	51.78	53.77	52.19	63.64	67.02	60.71	65.54	59.84	58.24	54.95	53.48	57.56
May	45.23	48.54	50.43	48.10	52.02	56.57	53.15	58.53	55.26	54.60	52.05	49.93	52.03
Jun	44.73	48.84	49.83	47.32	51.74	54.28	45.88	55.58	50.76	53.60	49.64	48.16	50.03
Jul	47.06	51.54	52.45	49.58	54.59	57.49	56.92	60.93	52.15	54.39	51.18	50.92	53.27
Aug	41.08	47.54	47.13	44.33	48.56	52.13	52.52	53.74	41.95	46.21	45.68	45.70	47.21
Sep	44.80	52.16	53.26	50.37	53.95	57.26	56.74	59.05	54.60	49.64	47.31	48.80	52.33
Oct	46.50	56.02	56.83	53.48	56.41	58.62	58.81	58.36	55.75	55.22	47.56	50.53	54.51
Nov	44.69	61.16	60.67	55.06	61.18	61.67	60.56	61.15	56.70	57.31	55.61	48.80	57.05
Dec	49.87	65.49	66.11	59.71	64.90	66.00	64.49	66.26	60.43	61.12	59.27	61.07	62.06
Mean	47.03	54.12	55.20	51.31	57.23	59.58	57.41	60.46	55.10	55.45	52.48	51.90	54.77

Table 10.13: Monthly returns (%) for combinations of buy and sell months for SXG

	Sell Month												
Buy Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	**Mean**
Jan	1.97	7.10	4.80	3.36	3.55	3.72	3.20	3.43	2.82	2.56	2.21	2.09	3.40
Feb	1.34	2.09	2.70	1.80	2.46	2.77	2.32	2.70	2.18	2.04	1.62	1.50	2.13
Mar	1.20	2.00	1.84	0.74	2.19	2.78	2.22	2.69	2.05	1.79	1.41	1.34	1.86
Apr	1.23	2.09	1.90	1.72	3.94	3.93	2.78	3.26	2.32	1.96	1.50	1.42	2.34
May	0.66	1.57	1.45	1.34	1.57	3.59	1.94	2.64	1.53	1.22	0.74	0.76	1.58
Jun	0.14	1.12	1.07	0.97	1.28	1.52	0.08	1.99	0.74	0.59	0.12	0.24	0.82
Jul	0.17	1.27	1.21	1.08	1.42	1.66	1.58	3.82	0.94	0.59	0.03	0.25	1.17
Aug	-0.85	0.51	0.59	0.52	0.92	1.23	1.16	1.48	-2.08	-1.17	-1.41	-0.68	0.02
Sep	-0.37	1.22	1.28	1.15	1.54	1.85	1.74	2.10	1.82	-0.92	-1.42	-0.30	0.81
Oct	0.26	2.18	2.18	1.87	2.22	2.52	2.33	2.72	2.33	2.13	-1.84	0.26	1.59
Nov	1.21	3.51	3.24	2.66	2.93	3.24	2.94	3.24	2.77	2.51	2.54	2.20	2.75
Dec	0.92	4.60	3.63	2.80	3.00	3.30	2.89	3.08	2.50	2.19	2.15	2.15	2.77
Mean	0.66	2.44	2.16	1.67	2.25	2.68	2.10	2.76	1.66	1.29	0.64	0.94	1.77

Table 10.14: Increase fraction (%) for combinations of buy and sell months for SXG

	Sell Month												
Buy Month	Jan	Feb	Mar	Apr	May	Jun	Jul	Aug	Sep	Oct	Nov	Dec	**Mean**
Jan	58.73	68.86	72.95	67.62	75.18	75.80	75.00	76.16	72.50	71.89	65.71	64.89	70.44
Feb	55.24	60.66	58.03	61.07	64.39	73.84	73.02	73.74	67.26	64.06	61.87	59.14	64.36
Mar	52.19	57.83	58.96	48.35	64.87	72.70	68.90	70.77	64.41	61.13	56.34	53.68	60.85
Apr	52.36	55.38	60.24	60.39	67.14	75.27	67.96	73.50	66.90	63.07	56.79	53.68	62.72
May	47.41	52.00	55.34	53.94	57.65	62.54	61.27	65.85	60.49	59.44	52.98	50.17	56.59
Jun	44.36	51.79	56.25	53.49	57.75	64.20	53.62	62.81	55.75	57.29	50.00	46.53	54.49
Jul	45.88	51.98	56.64	52.94	59.30	66.28	65.50	66.78	54.86	57.79	49.31	49.14	56.37
Aug	41.63	49.60	50.59	47.45	51.94	59.69	60.31	60.94	41.18	45.49	42.76	44.95	49.71
Sep	43.36	50.99	55.64	55.47	60.47	64.48	62.99	65.10	59.85	51.22	41.40	46.71	54.81
Oct	48.44	62.85	63.04	57.53	63.42	65.89	65.12	65.23	61.48	59.77	44.91	51.92	59.13
Nov	51.18	70.97	69.41	66.15	74.42	76.54	76.36	75.58	68.08	67.05	64.23	52.26	67.69
Dec	57.60	77.29	78.21	70.31	76.92	78.68	80.69	79.07	71.92	70.38	64.73	69.35	72.93
Mean	49.87	59.18	61.27	57.89	64.45	69.66	67.56	69.63	62.06	60.71	54.25	53.54	60.84

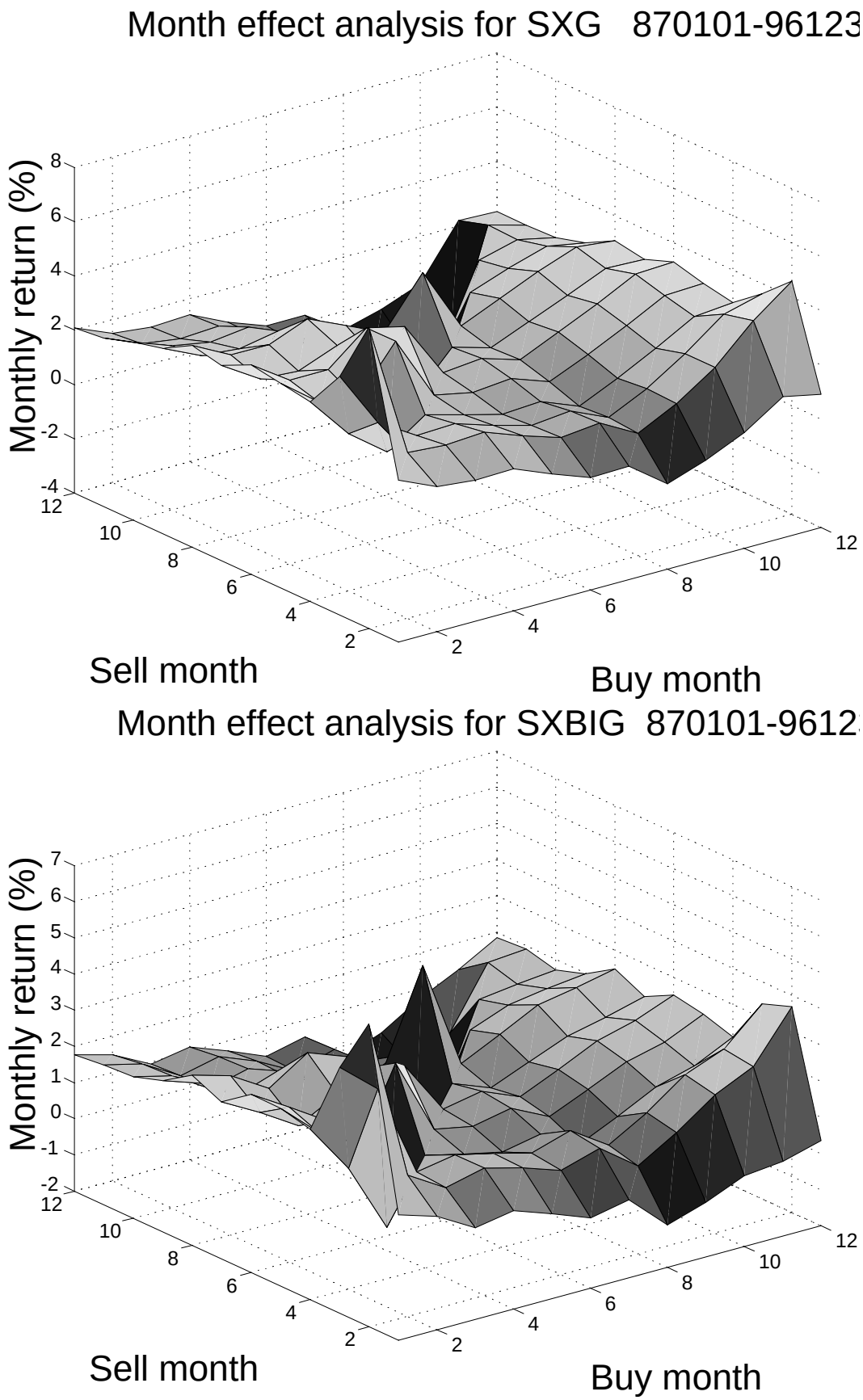

Figure 10.1: Monthly returns as a function of combinations of buy and sell months

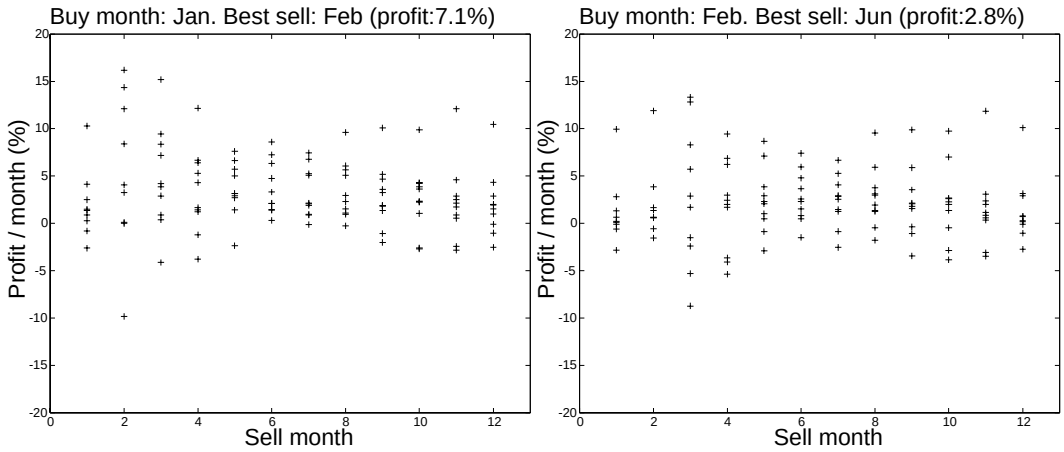

Figure 10.2: Monthly returns as a function of the sell month. Stocks: *SXG*. Years: 1987-1996.

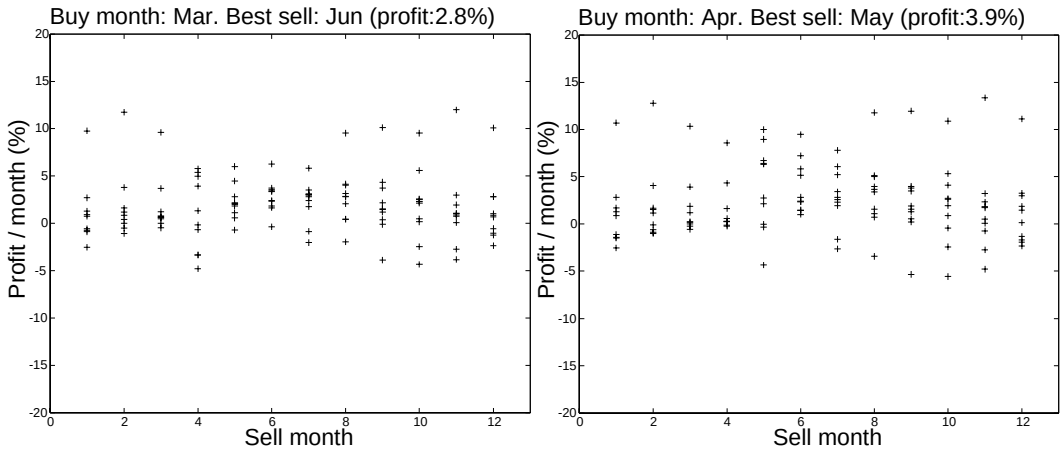

Figure 10.3: Monthly returns as a function of the sell month. Stocks: *SXG*. Years: 1987-1996.

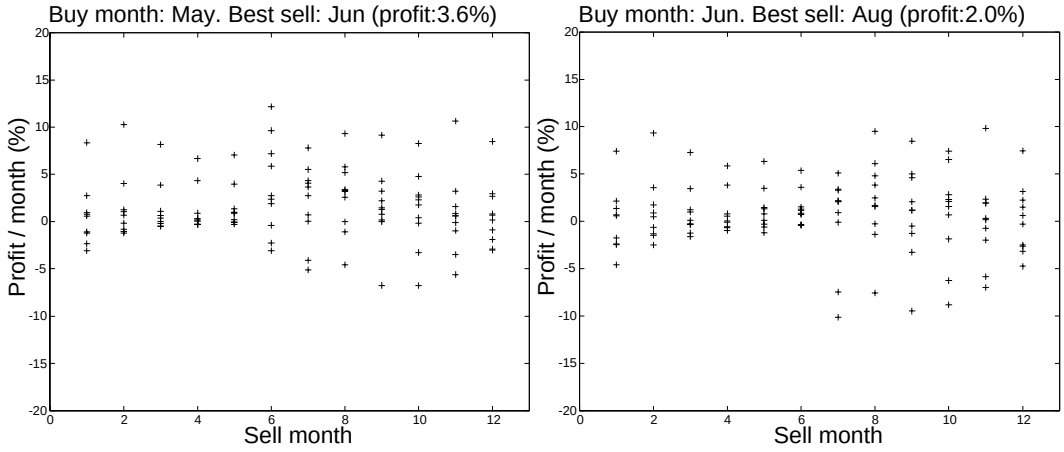

Figure 10.4: Monthly returns as a function of the sell month. Stocks: *SXG*. Years: 1987-1996.

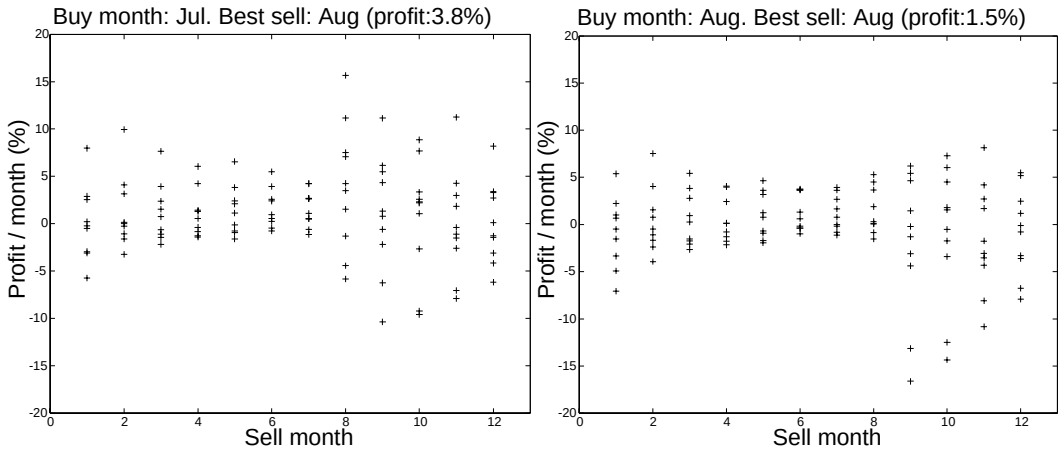

Figure 10.5: Monthly returns as a function of the sell month. Stocks: *SXG*. Years: 1987-1996.

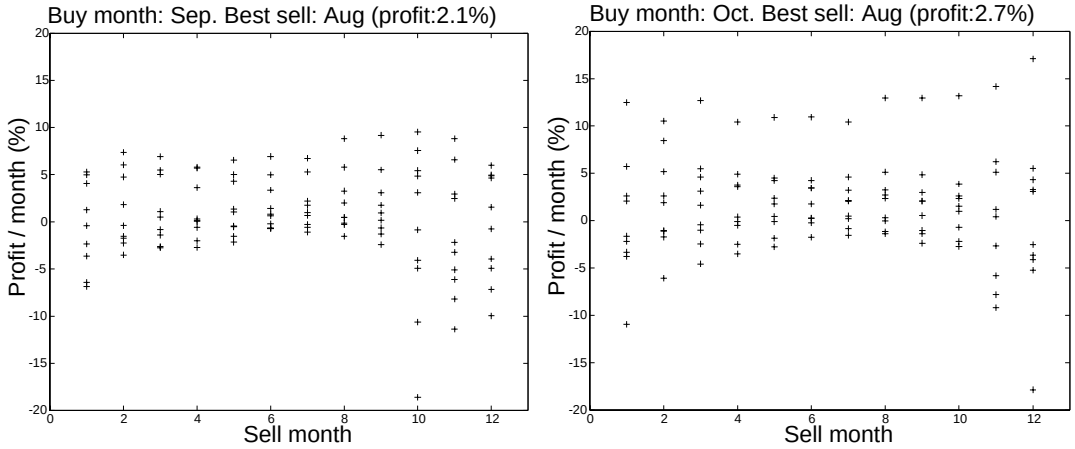

Figure 10.6: Monthly returns as a function of the sell month. Stocks: *SXG*. Years: 1987-1996.

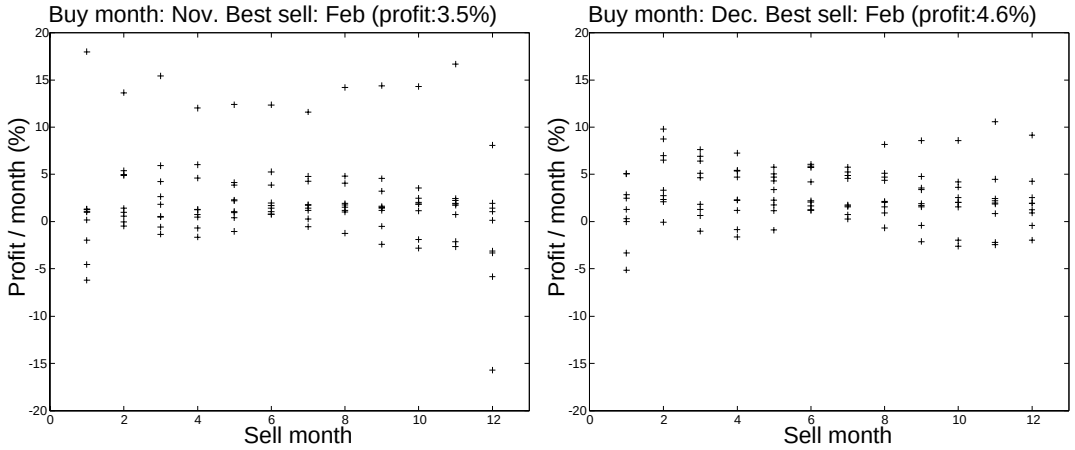

Figure 10.7: Monthly returns as a function of the sell month. Stocks: *SXG*. Years: 1987-1996.

Chapter 11

Ranks

The returns $R_k(t)$ (defined in (4.3)) are the primary target in most researches on the predictability of stocks. In this chapter we focus on the observation, that a real trading situation involves not only attempts to predict the individual returns for a set of interesting stocks, but also a comparison and selection among the produced predictions. We therefore introduce a rank concept A_k, based on the k-step return R_k as follows. The k-step rank A_k^m for a stock s_m in the set $\{s_1, ..., s_N\}$ is computed by ranking the N stocks in the order of the k-step returns R_k. The ranking orders are then normalized so the stock with the lowest R_k gets rank -0.5 and the stock with the highest R_k gets rank 0.5. The definition of the k-step rank A_k^m for a stock m belonging to a set of stocks $\{s_1, ..., s_N\}$, can thus be written as:

$$A_k^m(t) = \frac{order(R_k^m(t), \{R_k^1(t), ..., R_k^N(t)\}) - 1}{N - 1} - 0.5 \tag{11.1}$$

where the *order* function returns the ranking order of the first argument in the second argument, which is an ordered list. Thus, *order* returns an integer between 1 and N. R_k^m is the k-step returns computed for stock m. The scaling between -0.5 and $+0.5$ assigns the stock with the median value on R_k the rank 0. A positive rank A_k^m means, that stock m performs better than this median stock, and a negative rank means that it performs worse. This new measure gives an indication of how each individual stock has developed relatively to the other stocks, viewed on a time scale set by the value of k. The scaling around *zero* is also convenient when defining a prediction task for the rank. It is clear that an ability to identify, at time t, a stock m, for which $A_k^m(t + h) > 0$ means an opportunity to make profit in the same way as identifying a stock, for which $R_k(t + h) > 0$. A method that can identify stocks m and times t with a mean value of $A_k^m(t + h) > 0$, can be used as a trading strategy that can do better than the average stock. The hit rate for the predictions can be defined as the fraction of times, for which the sign of the predicted rank $A_k^m(t + h)$ is correct. A value greater than 50% means, that true prediction have been achieved. The following advantages compared to predicting returns $R_k(t + h)$ can be noticed:

1. The benchmark for prediction performance becomes clearly defined (refer to tables 8.2 and 8.3):

   - A hit rate $> 50\%$, for the predictions of the sign of ranks means, that we are doing better than the average stock. When predicting returns

$R_k(t+h)$, the general positive drift in the market causes more than 50% of the returns to be > 0.

- A positive mean value for predicted positive ranks $A_h(t+h)$ (and a negative mean value for predicted negative ranks) means, that we are doing better than the average stock. When predicting returns $R_k(t+h)$, the general positive drift in the market causes the returns to have a mean value > 0. Therefore, a mere positive mean return for predicted positive returns does not imply any useful predicting ability.

2. The rank values A_k^m for a set of stocks are uniformly distributed for all values between -0.5 and 0.5. Returns R_k are distributed with sparsely populated tails for the extreme low and high values. This makes the statistical analysis of rank predictions safer and easier than predictions of returns.

3. The effect of global events gets automatically incorporated into the predictor variables. The analysis becomes totally focused on identifying deviations from the average stock, instead of trying to model the global economic situation.

11.1 Statistical Properties

We start by looking at the rank variables as defined in (11.1). In Table 11.1 mean ranks $A_1^m(t+k)$ are tabulated as a function of $A_k^m(t)$. Table 11.2 shows the "$Up\ fraction$", i.e. The number of positive ranks divided by the number of non-zero ranks. Table 11.3 finally shows the number of observations in each table entry. Each row in the tables represents one particular value on k, covering the values $1, 2, 3, 4, 5, 10, 20, 30, 50, 100$. The label for each column is the mid-value of a symmetrical interval. For example, the column labeled **0.05** includes points with $k-$step rank A_k^m in the interval [0.00 0.10 [. The intervals for the outermost columns are open-ended on one side. Note that the stock price time series normally have 5 samples per week, i.e. $k = 5$ represents one week of data and $k = 20$ represents approximately one month.

Table 11.1: Mean 1-step ranks for 207 stocks

	k-day rank									
k	**-0.45**	**-0.35**	**-0.25**	**-0.15**	**-0.05**	**0.05**	**0.15**	**0.25**	**0.35**	**0.45**
1	0.067	0.017	-0.005	-0.011	-0.011	-0.004	-0.005	-0.010	-0.014	-0.033
2	0.060	0.017	0.002	-0.004	-0.010	-0.003	-0.007	-0.015	-0.017	-0.032
3	0.057	0.016	0.003	-0.005	-0.003	-0.008	-0.011	-0.011	-0.015	-0.034
4	0.054	0.018	0.003	-0.003	-0.005	-0.008	-0.011	-0.013	-0.012	-0.032
5	0.051	0.015	0.004	-0.002	-0.004	-0.009	-0.010	-0.009	-0.016	-0.032
10	0.040	0.013	0.005	-0.001	-0.003	-0.006	-0.007	-0.009	-0.012	-0.030
20	0.028	0.008	0.003	-0.003	-0.002	-0.002	-0.006	-0.011	-0.009	-0.019
30	0.021	0.007	0.002	0.004	-0.003	-0.003	-0.006	-0.006	-0.011	-0.015
50	0.014	0.005	0.000	-0.000	-0.001	-0.002	-0.005	-0.004	-0.006	-0.010
100	0.007	0.003	0.001	-0.002	-0.003	-0.004	-0.004	-0.004	-0.003	-0.008

To ensure that found patterns reflect fundamental properties of the process generating the data, and not only idiosyncrasies in the data, the relation between current and future ranks is also presented in graphs, in which one curve represents one year.

Table 11.2: Fraction up/(up+down) moves (%)

	k-day rank									
k	**-0.45**	**-0.35**	**-0.25**	**-0.15**	**-0.05**	**0.05**	**0.15**	**0.25**	**0.35**	**0.45**
1	59.4	52.9	49.1	47.3	48.0	49.6	49.5	48.2	47.8	46.4
2	58.4	52.8	49.7	48.9	48.4	49.7	48.9	47.4	47.6	46.2
3	58.1	52.4	50.3	48.9	49.1	49.0	48.1	48.1	47.8	46.1
4	57.5	52.5	50.4	49.2	49.0	48.7	48.0	47.9	48.6	46.3
5	57.1	52.0	50.4	49.4	49.1	48.5	48.2	48.6	47.7	46.3
10	55.6	51.7	50.4	49.8	49.3	48.8	48.7	48.5	48.2	46.3
20	53.8	51.1	50.2	49.6	49.4	49.5	49.0	48.3	48.7	47.8
30	52.7	50.9	50.3	50.8	49.1	49.2	48.8	48.9	48.5	48.4
50	52.0	50.7	49.6	49.9	49.6	49.6	49.0	49.3	49.1	48.9
100	51.4	50.4	49.9	49.5	49.2	49.2	49.0	49.2	49.6	49.1

Table 11.3: Number of points

	k-day rank									
k	**-0.45**	**-0.35**	**-0.25**	**-0.15**	**-0.05**	**0.05**	**0.15**	**0.25**	**0.35**	**0.45**
1	30878	30866	31685	30837	30434	31009	31258	30539	30951	31550
2	30926	30548	31427	30481	30442	31116	31263	30435	30841	31675
3	30922	30440	31202	30404	30350	31146	31061	30449	30814	31697
4	30887	30315	31052	30320	30371	31097	31097	30328	30777	31776
5	30857	30293	30951	30275	30191	31049	31144	30254	30701	31816
10	30755	30004	30648	29958	30004	30875	30889	30155	30571	31775
20	30521	29635	30306	29591	29679	30560	30580	29836	30377	31692
30	30388	29371	30083	29388	29567	30349	30437	29652	30190	31503
50	30117	29006	29728	28979	29306	29876	30109	29236	29927	31159
100	29166	28050	28790	28011	28238	29015	29049	28254	29012	30460

Figure 11.1 shows $A_1^m(t+1)$ versus $A_1^m(t)$ in the diagram on the left. I.e.: one-day rank the following day versus one-day ranks the current day.

The same relation for 100 simulated random-walk stocks is shown in the diagram on the right for comparison.

From figure 11.1 we can conclude that the rank measure exhibits a mean reverting behavior, where a strong negative rank in mean is followed by a positive rank. Furthermore, a positive rank on average is followed by a negative rank on the following day. Looking at the "*Up fraction*" in Table 11.2, the uncertainty in these relations is still very high. A stock with a rank $A_1(t) < -0.4$ has a positive rank $A_1(t+1)$ the next day in no more than 59.4% of all cases. However, the general advantages described above, coupled with the shown correlation between present and future values, do make the rank variables very interesting for further investigations. The rank measure can be used both as input and output in inductive learning methods such as neural networks, fuzzy rule bases, etc. In such cases, one could formulate modeling tasks like

$$A_1^m(t) = g(A_1^m(t-1), A_1^m(t-2), ..., A_1^m(t-k)) \tag{11.2}$$

where the function g for example is a feed-forward neural network. In this case, we attempt to predict the 1-day-rank of the current day by looking at the one-day-rank for the previous day, the day before, etc. Another possibility would be a model like

$$A_1^m(t) = g(A_1^m(t-1), A_5^m(t-1), ..., A_{10}^m(t-1)). \tag{11.3}$$

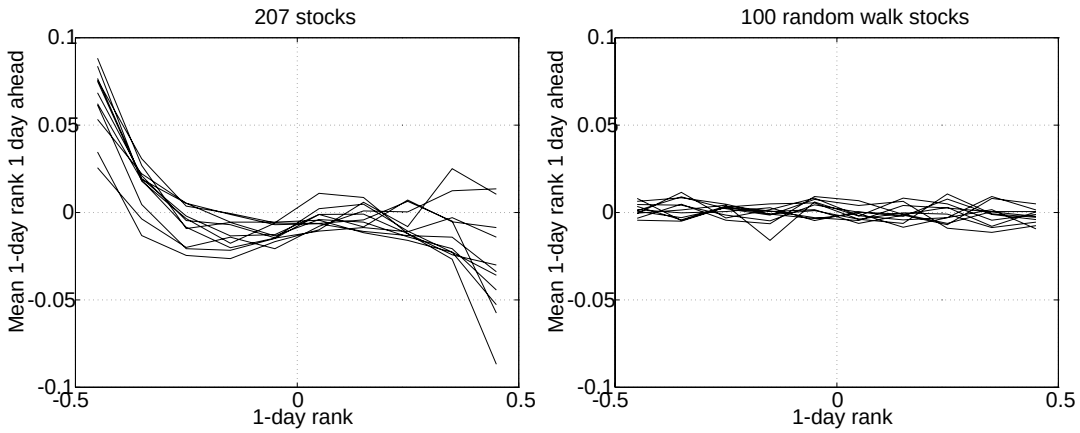

Figure 11.1: One-day ranks $A_1^m(t+1)$ versus $A_1^m(t)$. Each curve represents one year between 1987 and 1997. Real data in the diagram on the left and simulated random walk in the diagram on the right.

The one-day rank for the current day is predicted here from the 1-day rank, 5-day rank,...10-day rank, computed the previous day. The use of a neural network as a model function is, of course, arbitrary. Other choices would be fuzzy rule bases, radial-basis networks, or even linear models.

Chapter 12

Conclusions

We sum up the part of data mining with some of the most interesting observations.

- The investigations regarding autocorrelation show very low values with the possible exception of the first lag in the ACF. Surprisingly, the result for the 207 stocks in *SXBIG* shows a significantly *negative* correlation in the first lag, whereas *positive* correlation is found for the 32 major stocks in *SXG*.

- The analysis of trends produces supporting results. We observe that a 5% decrease in price since the previous day, stands a 61.2% probability of showing an increase by the following day. The cases with large *increase* since the previous day exhibit a difference between the two investigated sets of stocks. For the stocks in *SXG*, a 5% increase in price since the previous day, stands a 54.9% probability of showing an *increase* by the following day. For the stocks in *SXBIG*, a 5% increase in price since the previous day, stands a 53.3% probability of showing a *decrease* by the following day.

  However, the massive better part of returns falls into regions, where it is very difficult to claim *any* simple correlation between past and future price changes.

- The presented statistics show significant day-of-the-week and month-of-the-year effects on the stock returns. The proposed concept of daily returns R_D and a comparison to the ordinary step returns R_S, showed a significant difference, caused by the holiday effect. The daily returns R_D vary between 0.004% (buy Monday/sell Tuesday) and 0.243% (buy Thursday/sell Friday). The Increase Fraction I_D was shown to depend on the month of the year and to vary between 47.80% (March) and 54.44% (January). Even if the effects are too small to be utilized in actual trading, they are definitely big enough to influence other prediction algorithms, such as ordinary time series analysis or neural network models of daily returns. If not taken into account in such algorithms, the seasonal effects appear as a high-noise level in the data. It was shown, that the month-of-the-year effects are of the same size as the accuracy of many published prediction algorithms that do not make use of any date information.

  There are several ways to deal with the seasonal effects when constructing prediction algorithms:

  - Include the time dimension in the modeling, i.e. include a trainable parameter describing how the return depends on the day of the week, or on the month.

  - Aggregate data. For example, instead of modeling the return time series for all the days in a given time period, we can restrict the model to predict from one Monday to the next.
  - Without having done this statistical investigation, the most natural way to deal with the missing weekends might have been to expand the original time series $y(t)$ with interpolated data for Saturday and Sunday. However, the observed negative return between Friday and Monday shows, that such an approach can not be recommended.

- The rank concept has many properties that make it interesting to use as both inputs and outputs in prediction tasks.

Part IV

Methods

Chapter 13

Common techniques

13.1 Traditional Time Series Predictions

Take these broken wings and learn to fly
Blackbird, Lennon/McCartney

Traditional time series analysis might have begun in the end of the 20's, when Yule invented the autoregressive method (AR) 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:

$$y(t) = \sum_{m=1}^{d} a_m y(t-m) + e(t) \tag{13.1}$$

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

$$y(t) = \sum_{n=1}^{M} b_n e(t-n) \tag{13.2}$$

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 model (ARMA), which dominated the time series analysis for more than 50 years:

$$y(t) = \sum_{m=1}^{d} a_m y(t-m) + \sum_{n=0}^{M} b_n e(t-n) \tag{13.3}$$

An important step beyond these global linear models was taken by Tong and Lim [35] in 1980, when they suggested the *Threshold Autoregressive model* (TAR). TAR consists of two AR models, selected locally, depending on the system's state.

13.2 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 believed to have predictive power. We have conducted extensive tests with feature vectors, consisting of stock returns R_k defined in (4.3). For example, $p_t = (R_1(t), R_5(t), R_{10}(t), R_{20}(t))$. The results are presented in Chapter 15 together with an extension of the basic algorithm.

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. $R_k(t+k)$):

$$C_k(t) = \begin{cases} +1 & : \text{ if } \quad y(t+k) > y(t) \\ -1 & : \text{ if } \quad y(t+k) < y(t) \\ 0 & : \text{ if } \quad y(t+k) = y(t) \end{cases} . \tag{13.4}$$

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 exists a multitude of variants of the basic algorithm. A discussion of weighting schemes can be found in Robinson [29]. For an early work on time series applications and proofs of convergence, see e.g. Yakowitz [43].

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 is 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 14.3), may be more successful than the totally unbiased k-nearest-neighbors algorithm. One must also realize that k-nearest-neighbors 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 time series. 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.

13.3 Neural Networks

I didn't know what I would find there.
Another road where maybe I,
could see another kind of mind there
Got To Get You Into My Life, Lennon/McCartney

Neural networks are often 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, at least initially, 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. $y(t), y(t-1), y(t-2), \ldots$, 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 described in (13.1). The network output can be written as:

$$O(t) = g[y(t-1), y(t-2)..., y(t-d)], \tag{13.5}$$

  where g is a nonlinear function, created in the training of the network. As a special case $g(t) = \sum_{m=1}^{d} a_m y(t-m) + e(t)$ is recognized as an AR model. The motivations for this nonlinear generalization is the following. First, *Takens Theorem* [34] states, that for a wide class of deterministic systems, there exists a (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 $y(t) = g[y(t-1), y(t-2)..., y(t-d)]$. Second, it has been shown, that a neural network is a , capable of approximating any continuous function [9]. These statements together provide the basic motivation for the use of artificial neural networks in time series prediction.

- Recurrent 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 by the weights in the feedback loops of the network. Applications of recurrent networks can be found in Ellman [11].

- 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:

$$y(t) = \sum_{n=1}^{M} b_n x(t-n). \tag{13.6}$$

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

13.3.1 Neural Networks as Classifiers

In addition to being used for nonlinear regression, neural networks are often used for classification problems, where p-dimensional input vectors are to be mapped to an 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 an m-dimensional output vector, with values in the closed interval $[0, 1]$. It has been shown (see e.g. [17]), 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 so the zero-elements are represented by ε, and the one-elements 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.

13.4 Technical Stock Analysis

Oh, Carol, don't let 'em steal your heart away,
Well I'm gonna learn to dance if it takes me all night and day
Oh, Carol, C. Berry

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[1]. 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 so-called *Technical Indicators* for prediction of stock prices have been developed, and still are being developed on the grounds of these basic principles. Most of the developed prediction methods have a 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. In Section 5.1.3 we mentioned that most technical indicators can be described as *trading rules*, which produce buy and sell signals at certain times, as the price of the stock goes up and down. A thorough survey of the most common technical indicators can be found in [1]. The well-known *stochastics* indicator is analyzed in Section 16.5.

Some other examples are:

- Relative Strength Index (RSI). The relation between the average upward and downward price changes 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 as a sell signal.

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

- Moving average convergence divergence (MACD). Two moving averages with different backward windows are used. The difference between the moving averages is also smoothed. Buy and sell signals are generated from the crossings and trends of these filtered price signals.

Chapter 14

General Learning Issues

I've told you once and I've told you twice,
You better listen to my advice
(This could be) The Last Time, Jagger/Richards

In this chapter some general concepts relevant to the prediction tasks are surveyed. The focus is on the near-random-walk character of the stock prediction task.

14.1 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, stepwise 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. 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. These conditions are seldom fulfilled in real applications. The general limitations of statistical inference are also clearly understood, and formulated as the "bias/variance dilemma"[14].

14.2 Bias and Variance

To estimate an unknown function $f(P)$, which has produced a finite set of examples (P,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 $h_i(P)$ (e.g. a neural network)
3. Evaluate the estimator on a randomly picked test set of examples

Perform steps 1-3 repeatedly, producing hypothesis functions $h_1, ..., h_N$.

Now concentrate on what we have produced for a specific point p. We have a number of different estimators $h_1(p), ..., h_N(p)$. Since they all depend on random

selections of training data, they themselves can 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 p can be expressed as:

$$E_p = E[(h_P - f(p))^2] = E[h_P^2 + f(p)^2 - 2h_P f(p)] = E[h_P^2] + f(p)^2 - 2f(p)E[h_P]. \tag{14.1}$$

Inserting the identity $V_p = E[h_p^2] - m_p^2$ results in:

$$E_p = V_p + m_p^2 + f(p)^2 - 2f(p)E[h_P] = V_p + (m_p - f(p))^2. \tag{14.2}$$

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

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

The mean m_P can be estimated as $\sum_i h_i(p)/N$ and the variance V_P can be estimated as $\sum_i (h_i(p) - m_P)^2/(N-1)$. Note, that explicit estimation of E_P requires knowledge of the underlying function f. This is seldom if ever, the case, or 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 [10]. 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 $h_1(p), ..., h_N(p)$ produce roughly the same values, because a low complexity model does not have the power to express minor differences between different samples of training data. Hence the variance V_P is low. However, the bias is high, because all of the estimators differ in the same way from the true value $f(P)$.
- Too high complexity (e.g. a neural network with several layers and many weights) The computed estimators $h_1(p), ..., h_N(p)$ train precisely to each training set, thereby producing high variance V_P for the estimators, because each estimator operates on different random samples of training data. However, the bias is low, because the mean value m_P, computed as $\sum_i h_i(p)/N$, is centered around the true value $f(P)$.

Now turn to the "real" situation, where only one single estimate $h_k(P)$ has been produced. It is still an outcome of the random variable $h(P)$, 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 high-variance/low-bias? The choice is partly controlled by the complexity of the model. Looking at Equation (14.2), the correct answer would be, that neither of the alternatives is optimal in terms of producing 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 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 high-variance/low-bias. 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 $f(P)$. The bias then can be reduced without increasing the variance. This approach can be applied even to black-box models, as neural networks. In weight sharing several synapses (connections between nodes) use 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.

14.3 Weak and Strong Modeling

It ain't no use to sit and wonder why, Babe
Don't Think Twice, It's All Right, B. Dylan

The terms Weak and Strong Modeling refer 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 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 dynamic systems are sometimes modeled as fairly general nonlinear functions [7]. The classic ARMA models, described in Chapter 13.1, are also examples of weak models with few domain-specific assumptions.

14.4 Overfitting and Underfitting

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

14.5 Overtraining

You could have done better but I don't mind,
You just kinda wasted my precious time
Don't think twice, it's all right, B. Dylan

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 starts fitting properties in the training data that are not general, but rather to be considered as noise. The generalization performance therefore decreases from that point on in the learning process. The phenomenon 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) is less sensitive to the problems with overtraining. A high-noise level in data also increases the risk of overtraining, because noisy data gets interpreted more easily as real data by the training process. The risk of overtraining is also affected by the amount of data used for the training of the model. A huge amount 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" [42] , 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.

The problem with overtraining can be solved either indirectly, by reducing the model complexity, or directly, by interrupting the learning algorithm.

14.6 Selection Bias

The term Selection Bias is used to denote situations as the one described in Section 3.2, where the best performing method out of 100 candidates is selected. This selection process is identical to an overfitting situation as described above, where a much too complex model can be tuned to fit any data set. However, the selection bias is much more deceitful, since the non-selected methods disappear from the scene as soon as they are rejected. The only way to deal with the problem is to emphasize the proper use of a fresh test data set. However, in practice, one must realize that a selection bias is present in almost all presented research and results.

Bad results are simply not published. The random-walk nature of stock predictions makes this application very exposed to the problem. Selection bias of methods is, in my opinion, one of the hardest and most important problems to handle in the whole area of stock predictions.

14.7 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, 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 an important tool for model selection.

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

$$m \geq \frac{1}{\varepsilon}\left(\ln\frac{1}{\delta} + \ln|H|\right), \tag{14.3}$$

where ε is the error in a specific hypothesis and δ is the probability of an incorrect hypothesis, being consistent with all the 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 ε at most. Even if the practical results of PAC-learning are still limited, it focuses on two important issues:

- 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 the generalization ability.

A number of methods to estimate the generalization ability exist.

14.7.1 Test-Set Validation

The standard procedure of 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 have to be made regarding error distribution, or model linearity. 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, as architecture and initial weights. The method used to show this is a variation of the procedure described in Section 14.2. 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 regarded 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 also investigated the statistical difficulties with predictions of noisy data in Chapter 3.

14.7.2 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 ones are used to train a model. The performance of this model is estimated by the squared error in the sample left out. The procedure is repeated for all N samples in the data set, thus producing N models with associated error estimates for the single point left out. 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 [13]. A method specifically developed for artificial neural networks was proposed in [24]. 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.

14.7.3 Algebraic Estimates

The various methods of cross-validation require the data to be split to 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 exist, 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 (FPE):

$$FPE = MSE \cdot \left(\frac{1 + \frac{Q}{N}}{1 - \frac{Q}{N}} \right) \tag{14.4}$$

and the (GCV):

$$GCV = MSE \cdot \frac{1}{\left(1 - \frac{Q}{N}\right)^2} \tag{14.5}$$

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 therefore not sufficient to guarantee a low generalization error. The complexity is estimated by the number of free parameters Q.

However, 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.

Chapter 15

An Extended K-nearest-neighbors Analysis

Do onto neighbors,
What you do to yourself, yourself, yourself
Neighbors, Jagger/Richards

The method of k-nearest-neighbors was described in Section 13.2. In this chapter trend variables, as defined in (4.4), are used as input variables in a k-nearest-neighbors analysis, to find patterns of trend values that result in non-random future returns, as defined by (9.1). The k-nearest-neighbors algorithm is extended with a selection procedure, to find regions in the input space where the future returns are asymmetrically distributed. The new algorithm is successfully tested on artificial stock data, with trending patterns introduced. The algorithm is also applied to a number of national stock indexes: the American Dow Jones, the German DAX, the British FTSE, and the Swedish Generalindex. The results are positive for some of the tested indexes, and negative for others. Further tests must be conducted to attain statistically significant results.

15.1 Description of the Algorithm

A prominent property of stock-price time series is the high level of noise present. Thus, there is reason to seriously doubt the possibility of predicting future values, given only past values of the time series. What we can hope for is that the time series is sometimes predictable, and that in these situations it is possible to create a model that predicts the future better than mere chance. This approach is not acceptable in a general prediction situation, but is perfectly acceptable in the case of stock predictions. The performance of a trading system is normally calculated as the success rate, or generated profit, in the situations where buy or sell actions are suggested by the algorithm. The overall prediction accuracy is usually of minor interest. This is a natural fact for traders, and is implemented in the huge variety of technical indicators that issue buy and sell signals when certain conditions are met. A method, where a neural network is combined with a test for statistical dependence in the time series, can be found in [27]. We propose below an extension of the k-nearest-neighbors algorithm. Given the time series $\{y(t), t = 1, T\}$, the algorithm for prediction of the sign of $P_h(T)$ at time T looks as follows:

1. Generate patterns $\{p(t), t = 1, ..., T - h\}$. In the presented examples the patterns are defined as $p(t) = (T_5(t), T_{20}(t))$. Each pattern $p(t)$ is associated with a target value $P_h(t)$ as defined in (9.1). Also generate $p(T)$ with an unknown target value $P_h(T)$ to be predicted.

2. Compute the Euclidean distance between $p(T)$ and each of the patterns in $\{p(t), t = 1, T - h\}$. Select the k-nearest patterns and denote the set of associated target values Φ. Compute the homogeneity H of Φ as:

$$H = \frac{\max(\|\{x|x \in \Phi, x > 0\}\|, \|\{x|x \in \Phi, x < 0\}\|)}{\|\{x|x \in \Phi, x > 0\}\| + \|\{x|x \in \Phi, x < 0\}\|)}. \tag{15.1}$$

   The norm $\|.\|$ denotes the number of elements in the argument set. If the majority of elements in Φ is greater than zero, H is the fraction of elements greater than zero. If the majority of elements in Φ is less than zero, H is the fraction of elements less than zero. H is used as a measure of the degree of randomness in the target values Φ. A high value on H is interpreted as a high possibility to predict $P_h(T)$, using a k-nearest-neighbors method.

3. if $H \geq H_{\text{limit}}$, then the selected neighborhood around $p(T)$ is regarded as non-random, and the mean value of targets in H is used as predicted target value for $p(T)$. If $H < H_{\text{limit}}$, then the neighborhood around $p(T)$ is regarded as random, and a predicted target value for $p(T)$ is not evaluated.

Remark : Note, that the search for nearest neighbors in step 2 must not involve any data from the set $t > T$, because this data is highly correlated to the unknown value $P_h(T)$. That is the reason that the nearest neighbors are searched up to $t = T - h$ and no further. Using target values $P_h(t)$ where $t > T - h$ would involve peeping into the future beyond time T, which is the point where the prediction is calculated. Furthermore note, that the value on H_{limit} affects the number of situations where predictions are produced by the algorithm. The statistical significance of the performance therefore is reduced if too high a value for H_{limit} is chosen. In our tests, values between 0.6 and 0.9 have been used.

15.2 Generation of Test Data

Testing algorithms for stock data predictions is a difficult problem that requires extreme caution. The risk of interpreting random fluctuations as results of a successful prediction algorithm is, as we saw in Section 3.2, always present and sometimes surprisingly high. Therefore, to properly evaluate the developed algorithm described above, we construct an artificial stock time series. The purpose of the algorithm is to identify locations in the input space, where a correlation exists between input patterns and output $P_h(t)$. In all the examples in this report the prediction horizon h has the value 1. Since we use patterns with trend values, such as $p(t) = (T_5(t), T_{20}(t))$, as inputs, we introduce locations with correlation in a real stock index time series $y(t)$ by the following algorithm:

Compute the time series $T_5(t), T_{20}(t)$ *and* $P_h(t)$ as defined in Definition (4.4) and (9.1) using real stock data for $t = 1, .., N$.

1. *Repeat for* $t = 1$ *to* N

2. if $T_5(t) > 2$ *and* $T_5(t) < 4$ *and* $T_{20}(t) > 0$ *and* $T_{20}(t) < 5$ then
   % force a 1% increase in stock price after a positive 5-day trend
   % and positive 20-day trend :
   with probability 0.75 set $P_h(t) \leftarrow 1$
   end

3. if $T_5(t) > -4$ *and* $T_5(t) < -2$ *and* $T_{20}(t) > -5$ *and* $T_{20}(t) < 0$
   then
   % force a 1% decrease in stock price after a negative 5-day trend
   % and negative 20-day trend :
   with probability 0.75 set $P_h(t) \leftarrow -1$
   end

4. *next* t

In this way, the original time series obtains a correlation between the inputs and the output injected into two well-defined regions of the input space. It should be mentioned, that the introduced correlation is arbitrarily chosen just for illustrating the working of the algorithm. The regions in a real application are automatically detected by the algorithm. The deterministic points are few compared to the total number of generated points. For a 10-year time series (2500 data points) only 5% are typically generated in each of the two correlated regions in the input space. Such low signal to noise ratios is normally difficult to handle with methods like global regression analysis.

15.3 Simulation Results

In the two diagrams in Figure 15.1 the relation between the inputs $T_5(t), T_{20}(t)$ and the sign of the output $P_1(t)$ is illustrated. The top diagram shows the original data from the German stock index DAX. The lower diagram has correlation injected as described in the previous section. The input region $\{2 < T_5 < 4\ , 0 < T_{20} < 5\}$ was set to positive output with a probability of 75% and the input region $\{-4 < T_5 < -2\ , -5 < T_{20} < 0\}$ was set to negative output with a probability of 75%. The task of the prediction algorithm is to identify these regions and assign them to the correct class by the nearest-neighbors principle. As one can see in the bottom diagram, the correlation is not obvious even in this artificial case, where extremely high correlation is introduced into the data. In the case of the unmodified DAX data the task is even harder indeed, since no obvious regions with correlation can be seen in the top diagram. Figure 15.2 shows the situation when all the artificial data points have been run through the extended k-nearest-neighbors algorithm. The value of H_{limit} was set to 0.8 and the value of k, number of selected nearest neighbors, was set to 10. The figure denotes points, where the homogeneity was greater than 0.8 by crosses and the other points by dots. As can be seen in the diagram, the regions where correlation was injected are clearly identified by the homogeneity measure H. This

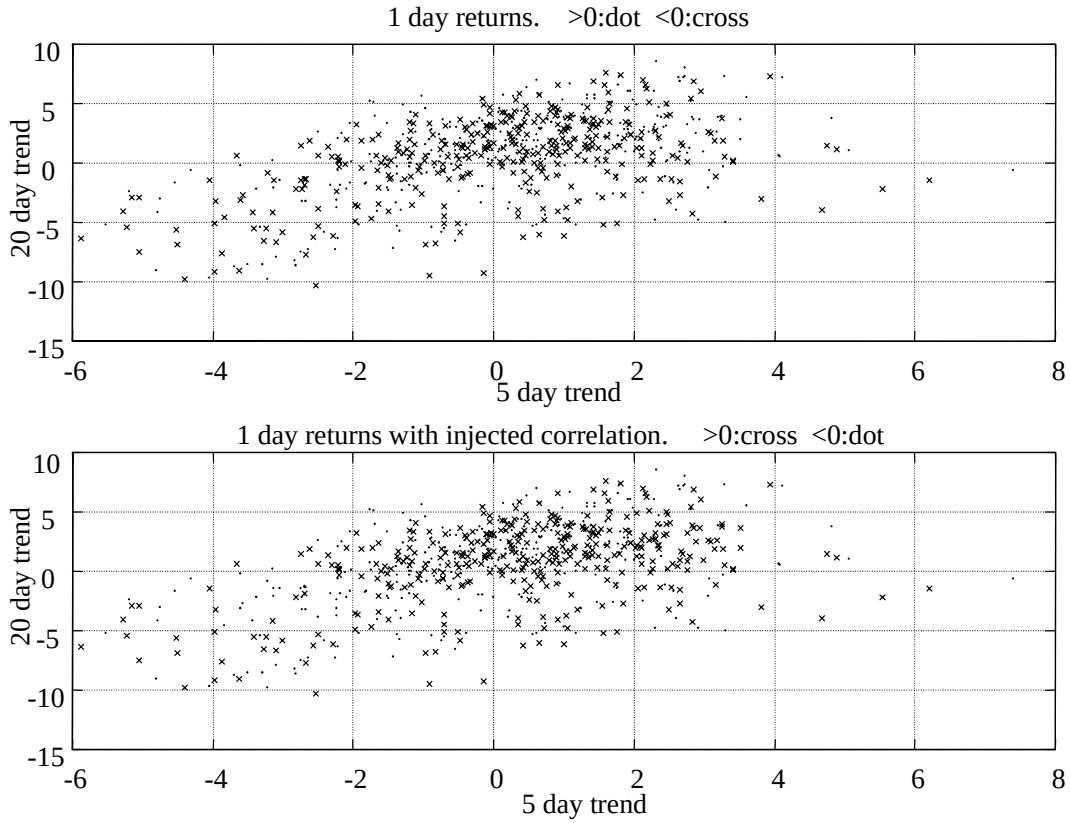

Figure 15.1: 1-day returns as a function of 5- and 20-day trends for DAX stock index

also shows up in the very high performance presented in Table 15.1. Figure 15.3 shows a similar diagram for the non-modified stock index DAX. Any clear regions with high homogeneity can be hardly identified.

15.4 Performance Analysis

Evaluation of prediction performance is an important, difficult, and often overlooked stage in the development of prediction algorithms for financial data. Since we are looking for very weak correlations, we always run a serious risk of interpreting random fluctuations in data as regularities with predictive power. The problem was further discussed in Section 3.2 and also in for example [21]. In Table 15.1, which presents performance measures for the predictions, we also present values for two benchmarks: Naive Prediction of Returns and the ϵ-increase prediction. The former is based on the assumption that next return will be the same as today's. The ϵ-increase prediction assumes that next stock price will increase by a very small amount ϵ since today. The reason that we do not use the usual zero-change benchmark instead of the ϵ, is that we want to compare hit rate performance to this benchmark model. Since the hit rate signifies the ability to predict the sign of the returns, we stipulate that the benchmark predicts an ϵ increase instead of zero. The benchmarks are used for calculation of the Theil coefficient T_R, HR_N and HR_ϵ. For definitions of the used concepts, refer to Section 6.3.

The investigated time period ranges from the beginning of 1987 until the end

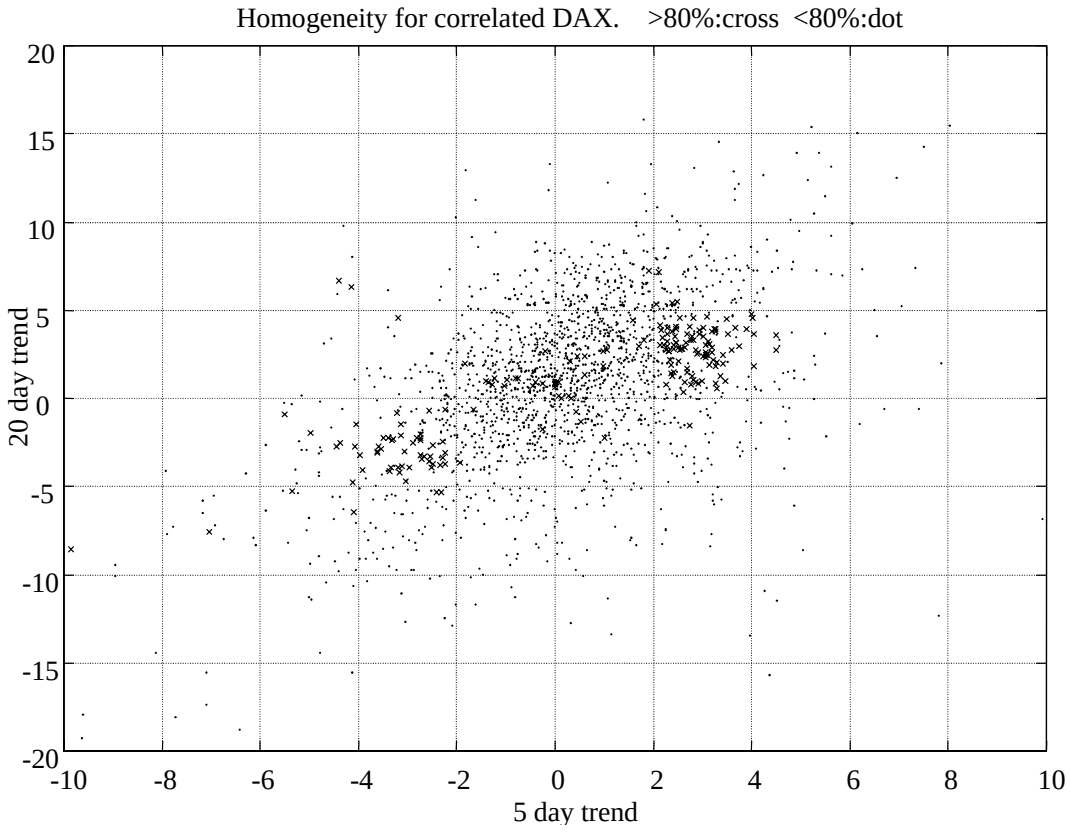

Figure 15.2: Homogeneity for correlated DAX. k=10.

of 1996. The limit of selection of points, H_{limit}, is set to 0.80 for all examples in this report. The parameter k, the number of selected nearest neighbors, is set to 10. The DAX column presents results of the real DAX index, whereas Correlated DAX presents results of the time series with injected correlation. As we can see from the first three lines, the performance for Correlated DAX is excellent. The hit rate 75.26% clearly outperforms both benchmarks. The Theil coefficient is also below 1, indicating a true predictive power beyond that of the ϵ-increase prediction benchmark. The algorithm produces 197 predictions, i.e. 10% of the total number. This is approximately the number of points affected by the injection of correlation, described in Section 15.2. The hit rate 75.26% also conforms nicely to the 75% randomness in the algorithm. The results for the real DAX index data appear to indicate predictability, even if the performance is clearly much lower than in the previous column with artificial data. However, one must bear in mind that the statistics are based on 45 selected data points only. The risk of data snooping is huge. The last three columns present prediction results for three other international stock indexes: the Swedish Generalindex, The American Dow Jones, and the English FTSE 100. The results are somewhat contradicting, and a statistically significant conclusion can hardly be made based on these tests only. However, the performance for the modified algorithm compared to the unmodified k-nearest-neighbors algorithm, shows a clear superior performance for the former. The Theil coefficient is significantly lower in four of the five columns for the modified algorithm.

An important parameter that one can alter in the k-nearest-neighbors algorithm

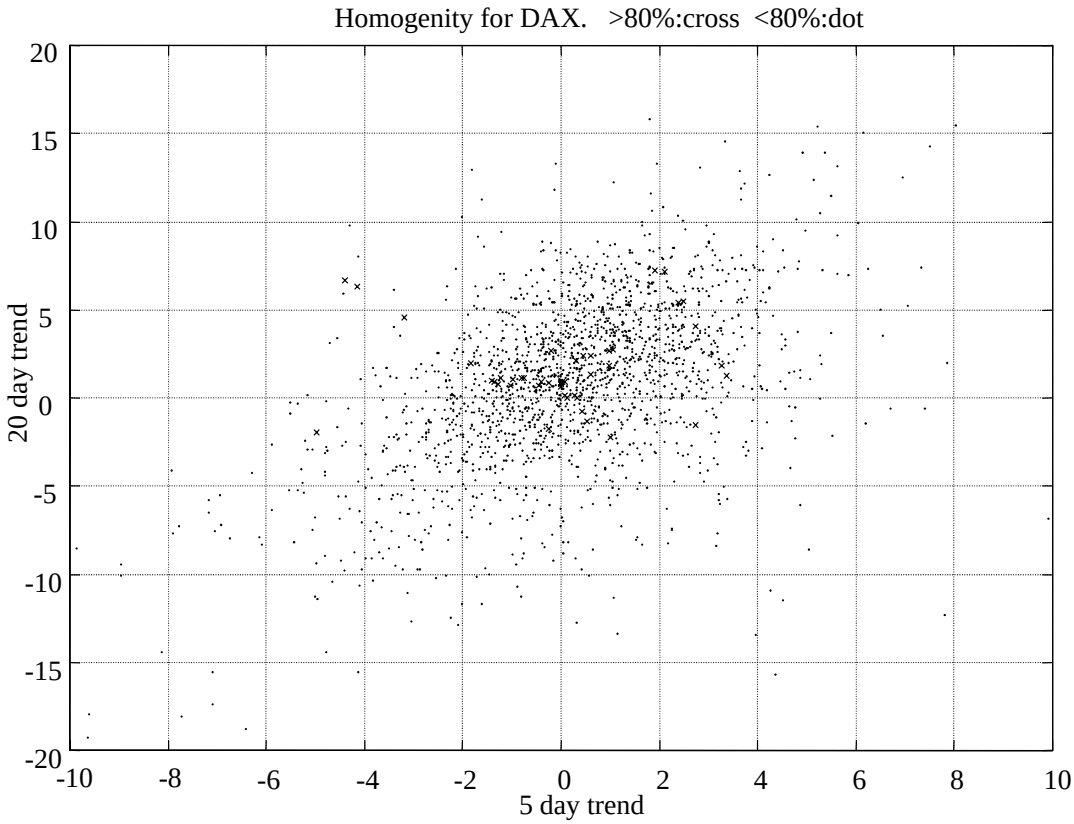

Figure 15.3: Homogeneity for DAX. k=10.

is the value on k. Table 15.1 shows results of $k = 10$. A thorough test with varying values on k has been conducted for the DAX and for the other reported stock indexes. The relative hit rate varies but cannot be shown to exceed 1 significantly for any value on k.

15.5 Conclusions and Further Development

The algorithm works well on the synthetic data with correlation injected, and also finds predictable patterns in the real stock indexes. However, the results are somewhat weak and further tests have to be conducted to obtain statistically significant results. Future work could include investigating higher dimensional patterns of trend variables, such as $p(t) = (T_1(t), T_2(t), T_5(t), T_{20}(t))$, and also combinations of these with normalized volume values to test the hypothesis (see e.g. [20]) that patterns in traded volume bear relevance to predictions of future returns. The size of the neighborhood could be determined in a more intelligent way than the fixed value of k neighbors used by the basic k-nearest-neighbors algorithm. Since the input space with trend patterns, such as $p(t) = (T_5(t), T_{20}(t))$, is not populated in a homogenous fashion, the k nearest neighbors may be picked in some areas of the input space from a very small volume, whereas in other areas, the k nearest neighbors have to be picked from a large volume, where the input space is sparsely populated. The suggested algorithm has natural application in areas other than stock predictions. It addresses all situations, in which predictability can be only expected to apply in

small and indeterminate regions of the input space.

Modified KNN	Corr. DAX	DAX	Gen.index	Dow Jones	FTSE
$HR_N = H_R/H_N$	1.40	1.14	0.92	1.02	1.02
$HR_\epsilon = H_R/H_\epsilon$	1.42	1.10	0.98	0.93	0.96
Theil coefficient T_R	0.93	0.78	1.00	0.86	1.13
$RMSE_R$	1.03	0.86	1.15	0.96	0.91
Hit rate H_R (%)	75.26	57.14	52.85	50.46	50.68
Number of points	197	45	125	123	74
Mean($\|predictions\|$)	0.79	0.52	0.58	0.54	0.49
Mean homogeneity H	66.03	62.26	65.82	63.46	64.11
Mean($\|returns\|$)	0.79	0.76	0.77	0.64	0.61
Naive pred.					
$RMSE_N$	1.54	1.58	1.49	1.64	1.10
Hit rate H_N (%)	53.80	49.95	57.72	49.63	49.86
Number of points	1930	1930	2317	2218	1848
ϵ-increase					
$RMSE_\epsilon$	1.10	1.10	1.15	1.12	0.81
Hit rate H_ϵ (%)	53.05	52.16	54.08	54.25	52.68
Number of points	1930	1930	2317	2218	1848
Standard KNN					
$HR_N = H_R/H_N$	1.04	1.01	0.94	1.02	1.02
$HR_\epsilon = H_R/H_\epsilon$	1.05	0.97	1.00	0.93	0.96
Theil coefficient T_R	1.02	1.05	1.04	1.08	1.06
$RMSE_R$	1.13	1.15	1.19	1.21	0.85
Hit rate H_R (%)	55.77	50.34	54.33	50.71	50.63
Number of points	1929	1929	2316	2217	1847

Table 15.1: Prediction performance for the *modified k-nearest-neighbors algorithm*, the *naive prediction of returns*, the $\epsilon-$*increase predictions* and a *standard k-nearest-neighbors algorithm*. Time period 1987-1996.

Chapter 16

ASTA - a Test Bench and Development Tool

So give me all your money,
give me all your gold
Some Girls, Jagger/Richards

16.1 Introduction

The idea of expressing stock prediction algorithms in the form of trading rules has gained considerable attention in academic research in the last years. The international conference NNCM-96 devoted a whole section in the proceedings to "Decision Technologies." Bengio [5] writes about the importance of training artificial neural networks with a financial criterion, rather than a prediction criterion. Moody and Wu [26] use reinforcement learning to train a trading system with objective functions, such as profit, economic utility, and Sharpe ratio. Atiya [3] describes a trading system based on time-variable, stop-losses, and profit objectives.

The general method with trading rules was described in Section 5.1.3. This chapter describes the system ASTA, which is an implementation of an Artificial Stock Trading Agent. With ASTA, trading-rule-based prediction algorithms are easily evaluated, using historical data. The trading simulation extends the single stock prediction problem to a multi-stock problem. In this way, the prediction problem becomes focused on relative performance rather than increase or decrease of individual stocks. Besides being an evaluation tool for existing algorithms, the system is a high-level development tool, where trading rules can be developed, combined, and tuned.

The program is written in the MATLAB programming language, and is used either as an ordinary objective function called from a user's program, or as a Windows-based tool for making benchmarks, and developing trading algorithms.

ASTA performs a simulation of multi-stock trading, where trading rules are executed for a large number of available stocks every day in the simulation period. The situation is fundamentally different from the single-stock prediction algorithms described in the previous chapters.

16.1.1 Objectives

The development of ASTA was instigated by a need for good working tools for the following research tasks:

1. A test bench for trading algorithms.

   Many technical indicators for stock prediction are accepted and widely used, without ever having been subject to an objective scientific analysis with historical stock data. It is true, that many commercial software packages for technical analysis offer both a comprehensive programming language, and a simulation mode, where the performance can be computed. However, most available products do not take this task very seriously and real trading simulation with a multi-stock portfolio is seldom possible. Furthermore, often the performance measures are not sufficient for a serious evaluation of the behavior of an algorithm for a longer period of time. Therefore, there is a need for a scientific test bench for the methods and algorithms already developed and in common use.

   The need for proper evaluation of new trading algorithms is of course the same as for existing ones. ASTA is developed in MATLAB and therefore is suitable for tests of algorithms developed in the same language, but MATLAB can also communicate with other languages.

2. An interactive development tool for trading rules.

   There are reasons to believe that a successful trading system consists of many disjunct parts, where a buy signal can be for example, "screened" by looking at the traded volume. A buy signal issued with a low traded volume may be then rejected. Other composite rules include looking at the general trend of the stock before accepting a signal from the system. ASTA provides the possibility to test such composite rules easily. The included function library and the possibility to define a trading strategy interactively, make implementation and evaluation of many trading strategies possible "without programming."

3. A non-interactive development tool for trading rules.

   Furthermore, it is possible and maybe also fruitful, to automate the development of trading rules. Since ASTA defines the trading strategy as symbolic Buy rules and Sell rules given as arguments to the system, it would be perfectly possible to construct buy and sell rules in a genetic framework, for example.

   Even if the general look of the algorithm is fixed, there are often a lot of tunable parameters that affect the trading performance. Examples are filter coefficients, order of polynomials and levels above or below which an entity should pass in order to generate a trading signal. Since we believe that the actual behavior during a realistic trading situation is essential for proper selection and optimization of an algorithm, there is a need for an objective function that can be included in an optimization phase for parameter tuning.

4. A data generating tool for post processing

   The comprehensive and user-friendly macro language in ASTA makes it a very suitable tool for extracting data for further analysis, such as classification with

neural networks or fuzzy rule bases. A raw selection of trading situations is first set up with simple trading rules. Data ("features") for these situations is then automatically written to a file. A "target" value for each trade is also supplied. Thereafter it is a classification task to find out how to distinguish between a profitable trade and a non-profitable one, based on the given features.

16.2 Design of the Artificial Trader

In this section we discuss the design and implementation of the ASTA system. The task of the Artificial Trader is to act on an artificial market with a large number of available stocks that vary in prices over time. The Trader has to execute the trading rule $T(t)$ at every time step, and decide whether to buy or sell stocks. The sole aim of the Trader is to produce as high a profit as possible. Various aspects of the calculation of the performance were discussed in the Section 16.2.1. The presentation and evaluation of the trading results are a major part of the ASTA system.

16.2.1 Performance Evaluation

The result of the artificial trader is presented as annual profits together with the increase in index. The mean difference between these two figures constitutes the net performance for the trader. The performance is displayed in both tabular and graphical formats as shown in the table part of Figure 16.3 and in Figure 16.4. The second last line of the table presents the annual profit above the index. The mean value of this entity is presented in the second rightmost column and represents a one-figure performance measure. However, the individual figures for each year should also be considered.

We now turn to the general architecture of the developed system.

16.2.2 Basic Architecture

The architecture of ASTA is based on an object-oriented approach with two major objects; the Market and the Trader. The two objects have a number of attributes and operations that can be applied on the objects. The basic components and their relations are presented in Figure 16.1.

The Market Object

The Market Object essentially consists of the total number of stocks that should participate in the trading simulation. A stock is defined by four time series; Close, High, Low, and Volume. Each of these time series has a numeric value (or NaN in the case of a unavailable value) for each date in the time period of interest. The basic operation on the Market Object is the simulation of changing prices as the date moves from start date to end date. The attribute T is updated by the Step-in-time operation applied to the Market Object.

The Trader Object

The Trader Object is more complex than the Market Object, as far as both attributes and allowed operations are concerned. The Portfolio attribute keeps track of the possession of stocks during the simulation. The Cash attribute is initialized to a certain value and thereafter is modified automatically as stocks are bought and sold. The Buy rule and Sell rule are the main attributes that affect the behavior of the Trader. They are expressed in a high-level language and may include calls to a large number of predefined MATLAB functions that access the stock data in the Market Object. User-defined functions can also be called directly. The values of the Buy rule and Sell rule attributes are set interactively to enable fast experimentation when developing trading algorithms.

The basic operations on the Trader Object are Buy_recommendations that evaluates the Buy rule and Sell_recommendations that evaluates the Sell rule. The result is vectors with buy and sell recommendations for all relevant stocks. These recommendations are then carried out by the Buy and Sell operation. The behavior of the Trader is modified by a number of Other parameters, e.g. minimum and maximum values for one individual trade.

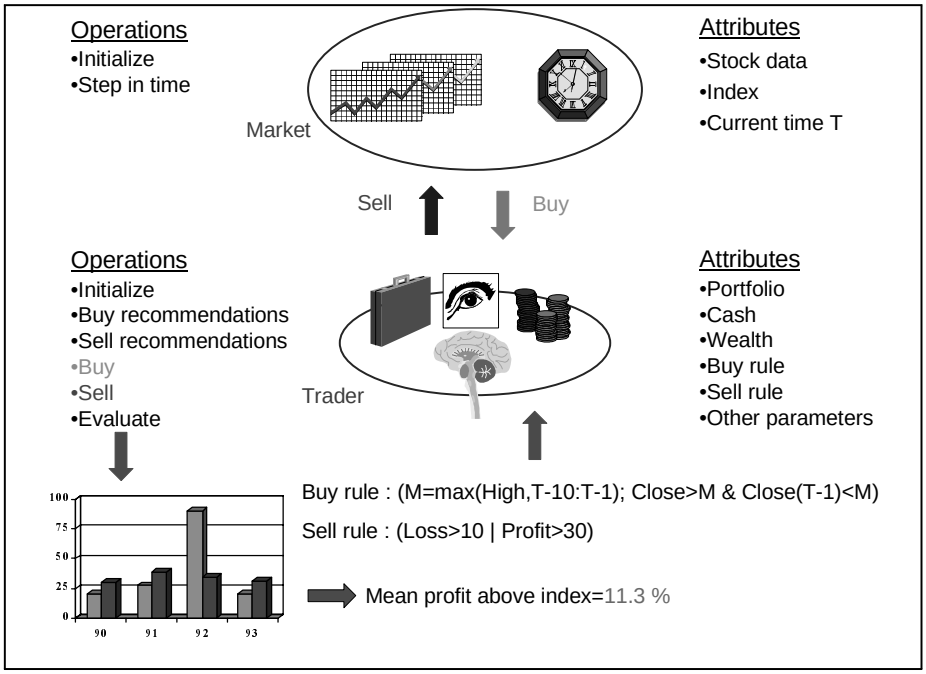

Figure 16.1: Basic Components of the ASTA System

Other Parts of the System

The Market and Trader Objects have to be controlled by a support system that takes care of the following "meta" operations:

- Simulation.

  The Step-in-time operation has to be applied to the Market Object in a loop for the selected time period. For each time step, the Trader Object also should be activated. The following pseudo code describes the full ASTA system:

```
Trader.Initialize
Market.Initialize
loop until Market.T≥EndDate
    s = Trader.Sell_Recommendations
    Trader.Sell(s) % Sell all stocks of type s
    s = Trader.Buy_Recommendations
    n = T.available_cash / length(s)
    Trader.Buy(s, n) % Buy n stocks of type s
   Market.Step in time
end loop
Trader.Evalute
```

- User interface.

  The end user assigns values to parameters such as the Buy rule and Sell rule of the Trader and the desired time period for simulation. After the simulation the computed performance of suggested trades is presented and optionally printed to different media.

  The "silent mode" makes it possible to use the system as an objective function in a parameter optimization. The Artificial Trader is then called as a standard MATLAB function from the optimization program code, returning a function value for each set of input parameters.

ASTA is available in two versions:

- As a MATLAB function, fasta.m, that takes a Buy rule and Sell rule as parameters and returns the performance for a selected time period.
- As a Windows application to be run under MATLAB. The Buy rule and Sell rule and all other settings are controlled interactively and the results are presented on the screen.

16.2.3 Predefined ASTA Functions

ASTA has a large number of predefined functions that make it possible to express compound trading rules interactively (as Buy rules and Sell rules). They also provide the developer of new algorithms with basic database access functions, as well as some useful high-level functions. A complete description of the predefined functions in ASTA can be found in Section 16.4.

16.2.4 Other Design Issues

We conclude the general description of the design with some specific issues that must be considered in the design of an Artificial Trader.

When Is Data Available?

In a real trading situation on day T, the Close, High, and Low for day T are not available. However, it is common that prediction algorithms assume that this data is available, and can be included in the prediction of the following day's prices.

Furthermore, it is often assumed that the Buy and Sell recommendations from the trading system can be executed using the close prices for day T. Of course, in reality this is hard to achieve, since the stock market is closed by the time the data for the current day becomes available, not to mention it having been transferred to a computerized trading system. ASTA can be configured to execute the simulated trades, using prices from either day T or day T+1 for buy and sell. The data for day T is assumed to be available for analysis on day T.

How Many Stocks to Buy and Sell?

The trading rule $T(s)$ does not contain any guide as to how many stocks or how big a portion of the cash should be invested in the particular stock that gets a buy signal. It is obvious that this affects the performance of a trading system, and it is also clear that a risk estimate coupled with the trading signal could be of help. The present version of ASTA does not offer any sophisticated procedures for the selection of the number of stocks to buy. The agent simply divides all available cash equally among the stocks that get a buy signal (taking into account the upper and lower limits for one individual trade). When a stock gets a sell signal, all shares are sold.

Reinvesting Money

The Artificial Trader buys and sells stocks and hopefully increases the wealth during the simulated time period. In such a strategy there is a cumulative effect, since old profits are reinvested in future trades. The reinvesting is necessary if the trading results are to be comparable with the change in index, e.g. in equity diagrams such as presented in Figure 6.1.

Interest on Cash

The money not invested in stocks is assumed to yield interest in a bank account. The level of interest varies over time, and in our test database follows the daily Swedish three-month-bond interest rate (configurable).

Unused Buy and Sell Signals

One problem of the basic approach of evaluation by simulated trading is that a large portion of the buy signals are neglected due to lack of money for the artificial trader. Even sell signals are neglected, simply because the stock that issues the signal is not in the trader's portfolio. Tests show that more than 90% of the signals often are ignored in this way. This seriously affects the statistical basis of a performance analysis for the trading algorithm. A possible way to attack the problem is a top level loop with multiple runs and randomization of entities like:

- The starting date of trading
- The acceptance of buy signals
- The set of available stocks for trading

The present version of ASTA does not include any of the suggested improvements. Since the running time of one simulation of a 10-year trading takes in the order of 0.5-3 minutes on a 266 MHz Pentium PC, the problem needs proper analysis before any further development along these lines is conducted.

16.3 Using ASTA

ASTA is available in two versions:

- As a MATLAB function, fasta.m, that takes a Buy rule and Sell rule as in parameters and returns the performance for a selected time period.
- As a Windows application to be run under MATLAB. The Buy rule and Sell rule and all other settings are controlled interactively and the results are presented on the screen.

16.3.1 Fasta

The purpose of the "batch" version of the program is to provide an objective function that can be used for parameter tuning or automated generation of trading rules. This report does not describe that part of the system more than by showing the following example of how it can be used:

Example 1 *:*

```
% load the workspace with stock data:
load('astasxg')
% Simulate trading with Buy and Sell rules for 1990-1995.
% The performance is returned in the p variable.
p = fasta('Clos(T)>Clos(T-1)','Profit>20 | Loss>10',[],[90 95])
```

16.3.2 Wasta

ASTA can be also run as an interactive Windows application in the MATLAB system. In this section examples from the interactive version are demonstrated.

To start ASTA, run the wasta.m function from the MATLAB environment. The screen layout is shown in Figure 16.2. In this picture, the user can set up parameters for a simulated trading. The stocks of interest are picked from a large database in Metastock format. Stocks from the Swedish stock market have been used in the presented runs. 32 major stocks with active trading for the years 1987-1997 have been selected. Other databases could be interfaced to the system easily.

The most interesting items are the lines "Buy rule" and "Sell Rule". This is the place where the trading algorithm is decided. The rules follow MATLAB syntax and can include any of the large number of predefined functions or the user's own functions with new algorithms. The simulation of trading starts by clicking on the "Run" button. The results are presented in tabular form and in graphs as in Figure 6.1.

The command button "Sweep" initiates a whole series of simulations with different values on symbolic parameters in the Buy and Sell rules. The name of the parameters is given in the "Parameter" text box, and the values to be included in

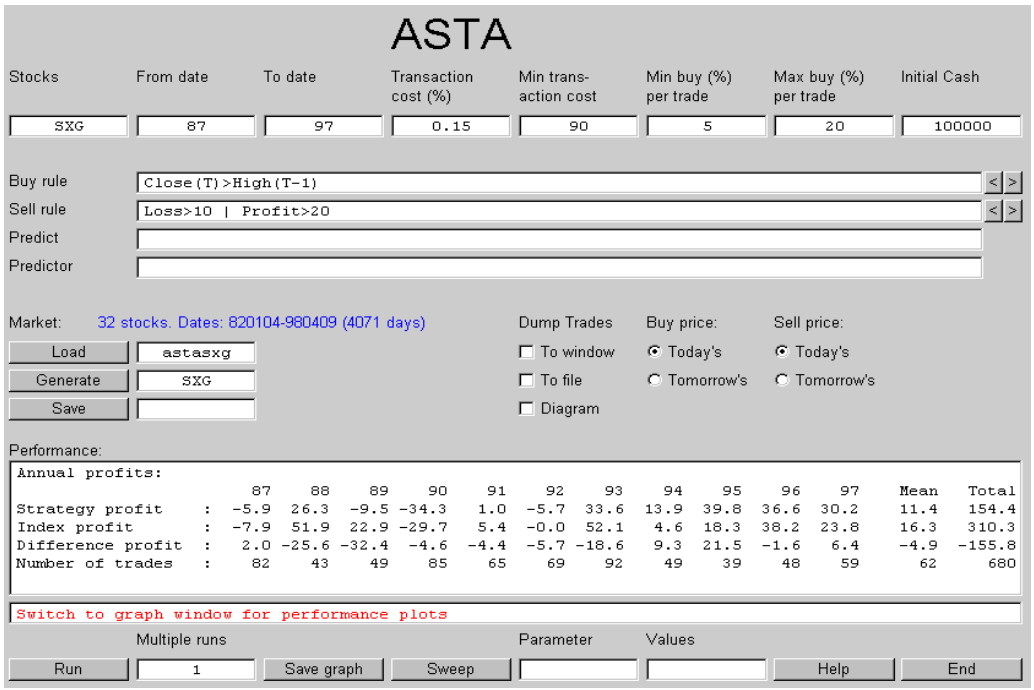

Figure 16.2: Screen layout for the windows version of ASTA

the multiple simulations are given in the "Values" text box. The results are presented in six graphs, which are also written as encapsulated postscript (eps) files. Examples can be found in Section 16.6.

16.4 Developing Trading Algorithms with ASTA

One of the key goals of using ASTA has been to provide a solid and easy-to-use basis for development of trading algorithms. The "Trading rule" approach has been taken as described in Section 5.1.3. The algorithm should be formulated in a day-by-day structure, where buy and sell recommendations are produced daily by the algorithm. The decisions should be based on past stock data only. Fundamental data about the companies cannot be used in the present version of the system. It should be also emphasized that the "Trading Rule" approach implies a decision-making system, and not a modeling system. However, implemented algorithms may use modeling "internally" and base the buy and sell recommendations upon it.

The following goals and guidelines have been kept in mind while designing the user interface in ASTA:

- Fool proofness

  An "off-by-one" error seldom results in more drastic consequences than when developing prediction algorithms. Looking into the future, which is normally pretty hard to do, is in the case of prediction algorithms surprisingly easy to do, even unintentionally. In the case of indexing time series vectors however, it should not come as a surprise that finding an error in code segments, like the following one, is indeed difficult.

$i = k - h + 1$

$x(m) = y(k + 2 * (j - j) - 1)...$

Therefore, the access functions implemented in ASTA have been designed with a check for arguments that peep into the future. The global variable T is automatically updated by the simulation mechanism to point to the "current day". Attempts to address the stock time series after this point, automatically issue an error message. Example:

$y = Clos(T - 4 : T)$ assigns the close prices for the current day and the 4 previous days to the variable y.

$y = Clos(T : T + 1)$ issues an error message when executed during the simulation.

The same check routine has been implemented in the higher level routines as well, thereby issuing clearer error messages pointing to the first instance where an illegal time reference is found.

- Ease of use

  Existing ASTA functions can be used interactively to create Buy rules and Sell rules. When the user has implemented a new trading algorithm, the MATLAB function can be used also in the Buy rules and Sell rules. It can be easily tried out interactively with different settings on parameters etc.

- High level functionality

  Predefined ASTA functions can perform operations, such as moving averages and detecting crossings between time series. Other predefined functions compute derived entities from the stock data time series: gaussian volume, volatility, trend and ranks. These functions can be included in the user-defined algorithms and simplify the coding considerably.

ASTA is equipped with a large number of predefined functions to be used interactively in Buy rules and Sell rules. They may be used also in user-defined functions. Before going into details of the available functions, some basic concepts are described.

16.4.1 Global Variables

The variables listed in Table 16.1 are dynamically set by the system and normally should not be changed by any user routines.

The variable T is central both when defining Buy rules and Sell rules and when writing new functions for prediction. The idea is that the user should never have to worry about the time aspect, since it is automatically taken care of by the simulation framework. By using T as index, the "current" day is always referenced. By using T-1, the previous day is referenced etc. Attempts to use T+1 issue an error message, thus prohibiting the program from peeping into the future.

It is seldom necessary to use the variable Stocks when writing user-defined functions. However, it should be made clear that the variable is set to all stocks in the market when the system looks for stocks to buy (i.e. when the Buy rule is evaluated) and to all stocks in the Trader's portfolio when the system tries to find stocks to

Name	Type	Size	Decription
T	numeric	1	Row pointer to "current" day
Stocks	numeric	$1xN$	Column pointers to allowed stocks. **Stocks** is set to all stocks in market when evaluating a **Buy rule** **Stocks** is set to all stocks in portfolio when evaluating a **Sell rule**
Market	struct	1	The **Market** object
Trader	struct	1	The **Trader** object
ST	struct	1	Parameters controlling the Trader's behavior

Table 16.1: Global variables in ASTA

sell (i.e. when the Sell rule is evaluated). The function call Clos(T) for example, returns a row vector with the close values for all stocks in the Stocks variable for the day T (the "current" day in the simulation).

16.4.2 Time Series Matrices

Most time series data within the ASTA system are enclosed in objects denoted Time Series Matrices, and are abbreviated TSM in this documentation. They are ordinary 2-dimensional MATLAB matrices, where each row represents one date and each column represents one stock. The actual dates and actual stocks in a particular matrix vary, but the last row most often corresponds to the "current" day T. The columns normally correspond to the stocks in global *stocks* variable.

The available data is the stock time series enclosed in the Market Object: Close, High, Low, and Volume. These are the only features available when creating Buy rules and Sell rules, and also when creating new prediction algorithms. Clos, High, Low, and Volume are the names of the access functions used throughout the system.

Example 2 *: The call* ***Clos(T-5:T)*** *returns a 6 row matrix, where each column is assigned the close value for a stock in the global* ***Stocks*** *variable. Row 1 corresponds to day* ***T-5****, row 2 to day* ***T-4,*** *down to row 6, which corresponds to day* ***T****.*

A whole range of functions that accept TSM's as arguments and/or generate it as output, are included in the system. This design principle turns out to be convenient, since the column wise matrix operations within MATLAB can be used, and makes the code very readable and effective. The MATLAB function OHIGH in the following example detects *Clos* values that cross a k day maximum of *High* from below.

Example 3 *:*

```
function b = OHIGH(k)
  C = Clos(T);
  Cm1 = Clos(T-1);
  M = max( High( T-k:T-1 )); % The call to High returns a k
                             % rows long Time Series Matrix
  b = C>M & Cm1<=M;
return
```

The function returns a binary row vector with a "1" in those columns where the corresponding stock fulfilled the condition. It is an example of the data type *Indicator Vector*.

16.4.3 Indicator Vectors

An Indicator Vector is a single-row TSM. Sell and Buy rules should evaluate to binary Indicator Vectors used directly as sell and buy recommendations. Other Indicator Vectors, such as the return values from the function calls Profit and Trend1, contain real numbers as values.

In the previous section the function OHIGH returned an Indicator Vector with binary values, "0" or "1". A number of predefined functions return Indicator Vectors, and a user-defined trading algorithm or component normally should return a vector of this type. This enables such compositions of Buy rules as:

$$\text{OHIGH(10) \& (Trend1>1 | Trend5>0.5)} \tag{16.1}$$

The Buy rule evaluates to a binary Indicator Vector with "1" for stocks where OHIGH(10) returned "1" and, where either the short trend is greater than 1%/day or the long trend is greater than 0.5%/day. Functions Trend1 and Trend5 are described in section 16.4.6.

16.4.4 Predefined ASTA Functions

This section provides a list of all predefined functions that have been included to make it possible to express compound trading rules interactively (as Buy rules and Sell rules) and also to provide the developer of new algorithms with basic access functions, as well as some useful high-level functions.

The functions are divided into four categories:

1. Feature Functions

   Feature Functions are primarily access functions to stock prices (Close, High, Low) and traded Volume. Numerous functions for commonly used derived entities, such as trend, gaussian volume, and volatility, are provided as well. A Feature Function always returns a TSM.

2. Indicator Functions

   Indicator functions return an Indicator Vector most often used in the logical expressions that define the Buy rules and Sell rules. Examples are standard technical indicators, such as MACD and Stochastics. Other Indicator Functions implement a dynamic stop-loss handling and achieved profit or loss since a stock was bought.

3. Operator Functions

   Operator functions perform a computation on one or several TSM's, and return another one. Examples are computation of moving averages, minimum values in time windows etc.

4. Utility Functions

   Some functions are support functions that may be of use primarily when debugging new algorithms. Functions that extract stock names and actual dates are in this category.

Note:

All functions of type 1, 2, and 3 return a TSM. Each column represents one stock in the global variable Stocks. Stocks is automatically set to all stocks in the market when the system looks for stocks to buy (i.e. when the Buy rule is evaluated) and it is set to all stocks in the Trader's portfolio when the system tries to find stocks to sell (i.e. when the Sell rule is evaluated). It is seldom necessary to use the Stocks variable explicitly in Buy rules or Sell rules or when writing user-defined functions.

16.4.5 General Parameters

This section describes some parameters, which are common to many of the predefined functions.

Days

This parameter tells for which day or days the function should return values. In the case of days being a vector, the result is a TSM with the same number of rows as elements in the days parameter. Each row contains values as if computed on the corresponding day. The default value for the days parameter is the global T, i.e. the current day.

Example 4 *: **Clos(T)** returns a single-row TSM with close prices for all the stocks in the global variable **Stocks**.*

Example 5 *: **Trend5(T-9:T)** returns a ten-row TSM with 5-day trends for all the stocks in the global variable **Stocks**.*

To simplify the description of functions below, the *days* parameter is not covered. The Description field in the tables refers to each row of the returned TSM.

Plot

Many functions have the optional *plot* argument. By setting this parameter to the global constant PLOT, the function is plotted versus time for all the stocks selected in a row. The exact contents of the plot depends on the function. User-defined functions can easily incorporate the plotting functionality, which is extremely useful during development and debugging.

Only one function at a time can have the *plot* parameter set to a non-zero value (the global constant PLOT equals to 1). The default value for the *plot* parameter is 0.

16.4.6 Feature Functions

Feature Functions are primarily access functions for obtaining stock prices (Close, High, Low) and traded Volume. Derived entities such as ranks and trends are also provided.

All the Feature Functions return a TSM with one row for each day in the *days* parameter and one column for each stock in the global *Stocks* variable. Table 16.2 lists all predefined Feature functions in ASTA. The *plot* argument is described in Section 16.4.5.

Function	Parameters	Description
Clos	**days, plot**	The close value
High	**days, plot**	The highest traded price during the day
Low	**days, plot**	The lowest traded price during the day
Volume	**days, plot**	The traded volume value
Rank1	**days, plot**	The 1-day rank value
Rank2	**days, plot**	The 2-day rank value
Rank5	**days, plot**	The 5-day rank value
Rank20	**days, plot**	The 20-day rank value
Trend1	**days, plot**	The 1-day trend (%/day)
Trend2	**days, plot**	The 2-day trend (%/day)
Trend5	**days, plot**	The 5-day trend (%/day)
Trend20	**days, plot**	The 20-day trend (%/day)
Gvol2	**days, plot**	The 2-day gaussian volume
Gvol5	**days, plot**	The 5-day gaussian volume
Gvol10	**days, plot**	The 10-day gaussian volume
Gvol20	**days, plot**	The 20-day gaussian volume
Volat2	**days, plot**	The 2-day relative volatility
Volat5	**days, plot**	The 5-day relative volatility
Volat10	**days, plot**	The 10-day relative volatility
Volat20	**days, plot**	The 20-day relative volatility

Table 16.2: Predefined feature functions

The following special concepts should be further explained:

K-Step Return

The k-step return $R_k^m(t)$ of a stock m with a price time series $Clos^m(t)$ is defined as:

$$R_k^m(t) = 100 \cdot \frac{Clos^m(t) - Clos^m(t-k)}{Clos^m(t-k)}. \tag{16.2}$$

By setting k at different numbers we get measures indicating how much the stock has increased since its value k days ago. The Return values are not pre-computed in ASTA. The related Trend functions described in the next section can be often used instead.

K-Step Trend

The k-step trend $T_k(t)$ of a stock m is defined as:

$$T_k^m(t) = \frac{1}{k} \cdot R_k^m(t). \tag{16.3}$$

By setting k at different numbers we get measures indicating how much the stock has increased per day since its value k days ago.

The four functions Trend1, Trend2, Trend5, and Trend20 can be used to access four return measures for $k = 1, 2, 5$, and 20. They are computed and stored during market initialization to speed up the use during the simulation phase.

K-Step Rank

The k-step Ranks A_k^m for stocks $\{s_1, ..., s_N\}$ are computed by ranking the N stocks by the k-step returns R_k^m. The ranking orders are then normalized, so the stock with the lowest R_k gets rank -0.5 and the stock with the highest R_k gets rank 0.5. Therefore, the definition of the k-step Rank A_k^m for a stock s_m, belonging to a set of stocks $\{s_1, ..., s_N\}$, can be written as:

$$A_k^m(t) = \frac{order(\ R_k^m(t),\ \ \{R_k^1(t), ..., R_k^N(t)\}\ ) - 1}{N - 1} - 0.5 \tag{16.4}$$

where the *order* function returns the ranking order of the first argument in the second argument, which is an ordered list. R_k^m is the k-step return (definition 16.2) computed for stock m.

The Rank for a stock is a measure seldom or never seen as input to automatic trading systems. Since the ultimate goal is to "beat the market", i.e. to do better than the average stock, it seems reasonable to have them available at least as potential inputs.

The four functions Rank1, Rank2, Rank5, and Rank20 can be used to access four rank measures for $k = 1, 2, 5$, and 20. They are computed and stored during market initialization to speed up the use during the simulation phase. The set $\{s_1, ..., s_N\}$ is all the stocks available for trading.

Gaussian Volume

The gaussian volume $V_n(t)$ is defined as:

$$V_n(t) = \ (V(t) - m_V(t))/\sigma_V(t) \tag{16.5}$$

where $m_V(t)$ and $\sigma_V(t)$ are computed in a running window of length n:

$$m_V(t) = \frac{1}{n} \sum_{i=1}^{n} V(t-i) \text{ and} \tag{16.6}$$

$$\sigma_V = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (V(t-i) - m_V(t))^2}. \tag{16.7}$$

V_n expresses the number of standard deviations by which the volume differs from its running mean. Therefore, it can be used as an indication of "abnormal" values,

caused by a sudden change in market interest for a particular stock. That is the reason that the sums in Definition (16.6) and (16.7) do not include the current day's volume, i.e. $V(t)$. The four functions Gvol2, Gvol5, Gvol10, and Gvol20 can be used to access four Gaussian volumes V_n for $n = 2, 5, 10$, and 20. They are computed and stored during market initialization to speed up the use during the simulation phase.

Relative Volatility

The Relative volatility $W_n(t)$ is here defined as:

$$W_n(t) = 100 \cdot \frac{\sigma_C(t)}{m_C(t)} \tag{16.8}$$

where $m_C(t)$ and $\sigma_C(t)$ are computed on the time series *Clos* in a running window of length n. I.e.:

$$m_C(t) = \frac{1}{n} \sum_{i=0}^{n-1} Clos(t-i) \tag{16.9}$$

and

$$\sigma_C(t) = \sqrt{\frac{1}{n-1} \sum_{i=0}^{n-1} (Clos(t-i) - m_C(t))^2}. \tag{16.10}$$

Definition (16.8) is recognized as the "coefficient of variation" or "relative standard deviation" in statistics. Relative volatility W_n is preferred to ordinary volatility estimation by the standard deviation, because it provides a measure that can be compared over longer time periods and also between stocks with totally different price levels.

The four functions Volat2, Volat5, Volat10, and Volat20 can be used to access four volatilities W_n for $n = 2, 5, 10$, and 20. They are computed and stored during market initialization to speed up the use during the simulation phase.

16.4.7 Indicator Functions

These functions return an Indicator Vector, i.e. a TSM with one row and one column for each stock in the global variable Stocks. Each column can be used in logical expressions to compose Buy rules and Sell rules. They can also be used with user-defined functions to create compound trading rules. Table 16.3 lists all predefined Indicator Functions in ASTA. The plot argument is described in Section 16.4.5. Some Indicator Functions require further explanation.

Profit and Loss

The system automatically keeps track of the change in price of each stock since it has been bought. The Profit is defined as:

$$\text{Profit}(t) = (\text{Clos(t)} \ / \ \text{BuyPrice} - 1) * 100. \tag{16.11}$$

The Loss is sometimes a more convenient measure and is defined as:

Function	Parameters	Description
Profit	-	% increase in **Close** since buy of stock
Loss	-	% decrease in **Close** since buy of stock
Stoploss	**S1, S2, P1**	Implements a dynamic stop-loss function: Stop loss is initially set to **S1** If **Profit>P1**, change the stop loss to **S2**, **Stoploss** returns "1" iff the **Loss** is less than the dynamic set stop loss. This function keeps track of each stock separately.
Underlow	**L**	"1" iff **Close** crosses a **L** day min of **Low** from above
Overhigh	**L,plot**	"1" iff **Close** crosses a **L** day max of **High** from below and **High(T-1)** is less than the same max
Days	**dys**	"1" iff day number is in vector **dys** (1:Monday...7:Sunday)
Stoch	**K,Ks,S,plot**	The technical indicator Stochastics.
Macd	**K,D,S,plot**	The technical indicator MACD

Table 16.3: Predefined indicator functions

$$\text{Loss}(t) = (1 - \text{Clos(t)} \ / \ \text{BuyPrice}) * 100. \tag{16.12}$$

BuyPrice is the average buy price of each stock in a portfolio. Profit and Loss are not defined for stocks outside the portfolio.

The functions are of great use when defining trading rules. For example, a simple Sell rule may be defined as:

$$\text{Loss}>10 \mid \text{Profit}>20 \tag{16.13}$$

The Sell rule evaluates to an Indicator Vector with "1" for stocks where either Loss returned a value less than 10 or Profit returned a value greater than 20. A more complex function based on the profit and loss concept is Stoploss.

Stoploss

An often mentioned "trading rule to live by" is that of never allowing a profitable trade to turn into a loss. This idea is implemented in the function Stoploss. Each stock gets a stop-loss parameter of its own. This parameter is initialized to the value S1 when a stock is bought. It is changed to S2 when the Profit function registers more than P1% profit. Typical values for the parameters S1, S2 and P1 are 20%, -5%, and 10%. This results in the following behavior during trading:

1. The stock is sold immediately if the price drops by more than 20% below purchase price, thus cutting the maximum loss.

2. If the price ever rises by 10% above the purchase price, a consecutive drop in price to 5% above purchase price issues a sell signal. In this way, a profitable trade never turns into a loss.

16.4.8 Operator Functions

These functions perform an operation on one or several TSM's and return another TSM with transformed values.

All Operator Functions return a TSM with one row for each day in the *days* parameter, and one column for each stock in the global Stocks variable. The parameters x, x1, and x2 can be either a TSM or a string with the name of a function returning a TSM.

The parameter days:

- If x is a TSM, the days parameter should be row numbers in the matrix. The default value for days is the last row in the matrix.
- If x is a string, the days parameter should be a vector with day numbers, such as T, T-1, etc. The default value for days is T.

Note:

The Operator Function performs its operation on each of the lines in the x matrix and returns a TSM with the result for each row. In the following description of Operator Functions, the d variable is implicitly assumed to run through the full range defined by the days parameter. Each value on d results in one row in the output TSM.

Table 16.4 lists all predefined Operator Functions in ASTA.

Function	**Parameters**	**Description**
Minn	**x, L1, days, plot**	**L1**-day-min value in the rows of **x**
Maxx	**x, L1, days, plot**	**L1**-day-max value in the rows of **x**
Mav	**x, L1, days, plot**	**L1**-day-moving av. for columns in **x**
Stdev	**x, L1, days, plot**	**L1**-day-st.deviation for columns in **x**
Mavx	**x1,L1,x2,L2,A, days,plot**	"1" iff **Mav(x1,L1)** crosses **Mav(x2,L2)** from below. The **A** argument (optional) is the min required angle for the crossing.

Table 16.4: Predefined operator functions

Minn

Computes the minimum value in an L1 days long window backwards.

If x is a string, $Minn$ is defined as:

$Minn = min(x[d - L1 + 1 : d])$, where the square brackets denote function invocation of x.

Example 6 $Minn('Clos', 4, T)$.
Will generate a 1-row TSM: $min(Clos(T - 3 : T))$.

Example 7 $Minn('Clos', 4, T - 1 : T)$.
Will generate a 2-row TSM: $\begin{pmatrix} min(Clos(T - 4 : T - 1)) \\ min(Clos(T - 3 : T)) \end{pmatrix}$.

If x is a not a string it should be a TSM. $Minn$ is defined as:
$Minn = min(x(d - L1 + 1 : d, :))$.

Example 8 *$Minn(Z, 4, 10)$ where Z is a 10-row-long TSM. Will generate a 1-row TSM: $min(Z(7 : 10\ , :))$.*

Example 9 *$Minn(Z, 4, 9 : 10)$ where Z is a 10-row-long TSM. Will generate a 2-row TSM:* $\begin{pmatrix} min(Z(6 : 9\ , :)) \\ min(Z(7 : 10\ , :)) \end{pmatrix}$.

Maxx, Mav, and Stdev

These functions have the same parameters and behavior as Minn above. Just substitute min for max, mean, and std.

Mavx

Crossings between moving averages of various signals are often used in the definition of traditional technical indicators. The Mavx function supplies a convenient way to implement the detection of such crossings. The optional argument A makes it possible to specify the minimum angle for the crossing. Crossings at angles lower than the specified one do not generate a logical "1" at that point. Mavx can be used to detect both crossings from below and above, by changing the order of L1 and L2 arguments.

Example 10 *Mavx('Clos',2,'Clos',20,45) returns a binary Indicator Vector with "1" iff a 2-day moving average of **Clos** crosses a 20-day moving average of **Clos** from below in an intersection angle no less than 45 degrees. The computation is performed "today" since the default value for the **days** parameter is **T**.*

16.4.9 Utility Functions

These functions are support functions that may be of use primarily when debugging new algorithms. They can not be included in Buy rules and Sell rules, because they do not return Time Series Matrices. Table 16.5 lists all predefined Indicator Functions in ASTA.

Date	**days**	Vector with the real date (yymmdd) for **days**
Stockname	**s**	String with the name of stock **s.**

Table 16.5: Predefined utility functions

16.5 Examples

This section presents one example from the Windows version of ASTA. 32 major stocks with active trading from the Swedish stock market for the years 1987-1997 have been selected for analysis. The main ASTA screen is shown in Figure 16.3. The fields "Buy rule" and "Sell Rule" are the places where the trading algorithm

is defined. The rules follow the MATLAB syntax and can include the predefined functions or the users' own functions with new algorithms.

The example shows a test of the Stochastics indicator $Stoch[\ t,\ K,\ KS,\ D,\ Buylevel,\ Sellevel]$, common in technical trading, and here defined as:

$$\begin{aligned} Stoch(t) &= mav(100 * (Clos(t) - L)\ /\ (H - L),\ D) \quad \text{with} \\ L &= mav(\min(Low(T-K:T)),\ KS) \quad \text{and} \\ H &= mav(\max(High(T-K:T)),\ KS). \end{aligned} \tag{16.14}$$

The parameters K, KS and D are the window lengths of the moving average function mav.

A Buy signal is issued when $Stoch(t) > Buylevel$ and a Sell signal when $Stoch(t) < Sellevel$. The parameters K, KS, D, $Sellevel$, and Buylevel control the performance of a trading strategy based on the indicator and are subject to analysis in the next section.

In Figure 16.3, the "standard" values $30, 3, 3, 20, 80$ are used for the parameters. The results shown in Figure 16.3 and Figure 16.4 are quite stunning, with an average annual profit of 53.4% compared to the 16.3% achieved by the index. It is noteworthy that the only negative result is for the year 1997, where the strategy only made 1.7% whereas the index increased by 23.8%.

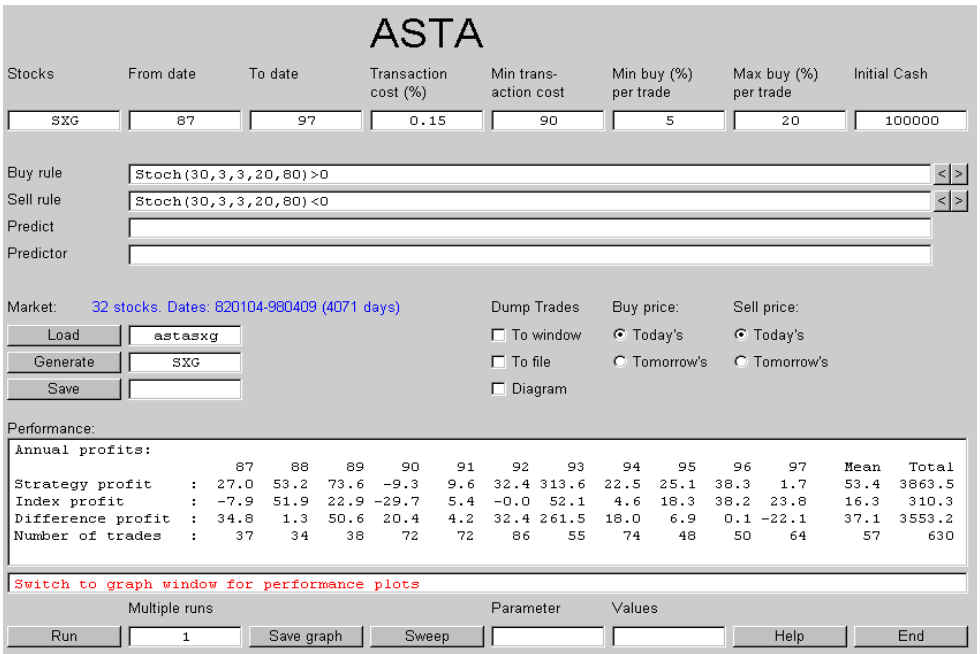

Figure 16.3: ASTA command window with Stochastics buy and sell rules

16.6 Viewing the Trader as an Objective Function

The obvious wish to maximize the profit of a trading system can be tackled in two ways:

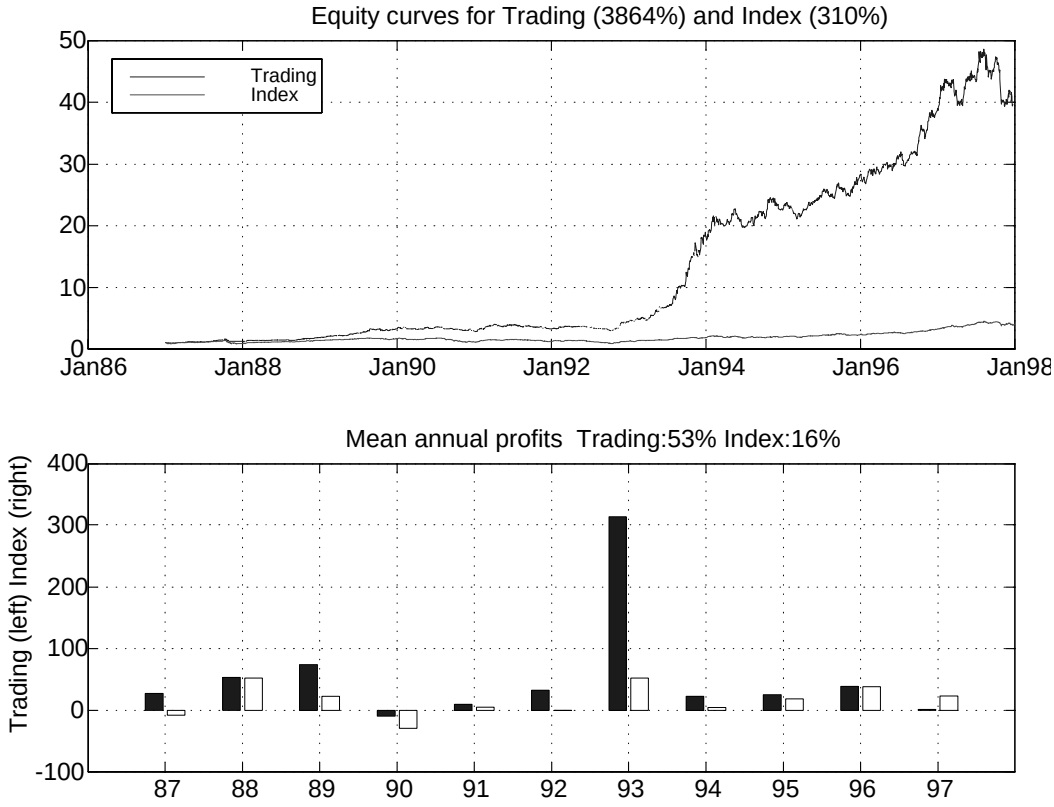

Figure 16.4: Performance of the Stochastics indicator.

1. Parameterizing the Buy rule and Sell rule, i.e. introducing parameters within the rules. Example:

   $Buy\ rule\ =$ '$Clos(T) > Maxx($'$High$', $Nhigh, T-1)$'

   $Sell\ rule\ =$ '$Loss > L \mid (Profit > P\ \&\ Clos(T) < Clos(T-1))$'

   The function Maxx returns the maximum time series value in the interval $[T - Nhigh, T - 1]$. It is now possible to optimize the profit P with respect to the parameters $Nhigh, L$ and P.

2. Viewing the Buy rule and Sell rule as symbolic expressions. The optimization then turns into a search problem, most naturally implemented in a genetic framework or using Inductive Logic Programming.

An optimization of one of the parameters in the Stochastics indicator serves as an example for approach 1. We set up an optimization with buy and sell rules according to:

$Buy\ rule\ =$ '$Stoch(30, 3, 3, Sellevel, 80)$'

$Sell\ rule\ =$ '$Stoch(30, 3, 3, Sellevel, 80)$'

The excess profit P can be optimized now with respect to $Sellevel$. The graphs in Figure 16.8 are automatically generated by the "Sweep" function in the ASTA system. The name of the parameter is given in the "Parameter" text box and the range in the "Values" text box.

It must be emphasized that optimization of the performance is a multi-dimensional parameter estimation problem. The graphs presented show the profit P as a function of *one* of the involved parameters, whereas the rest of the parameters are fixed.

The main purpose is to illustrate the possibilities and problems involved, even in a one-dimensional optimization.

From the summary graph (top left in Figure 16.8) over the entire training period, we can deduce that the highest profit is achieved for a *Sellevel* somewhere around 35. However, a look at the data at a higher resolution reveals a more complicated situation. In the bottom left diagram the same relation is plotted with one curve for each year in the training data set. It is clear that the mean profit is totally dominated by the results from one of the years (1993).

From these curves we can learn at least two important points:

1. The spread between individual years is very high.
2. The location of the maximum is not obvious.

This behavior of data is typical for most variables. The profit cannot be easily described as a function of measurable variables without introducing a dominant noise term in the function. Let us therefore view the annual excess profit as a stochastic variable $P(\theta)$, where θ stands for one particular setting of the parameters that affect the profit. In the shown example, θ is the *Sellevel* parameter. $P(\theta)$ has been sampled once per year during the eleven years in the data set:

$$\{P_1(\theta), P_2(\theta), P_3(\theta), P_4(\theta), P_5(\theta), P_6(\theta), P_7(\theta), P_8(\theta), P_9(\theta), P_{10}(\theta), P_{11}(\theta)\}. \quad (16.15)$$

Viewed this way, the task of tuning the parameter θ to find the "maximum" profit P is not well defined. P is a stochastic function and consequently has no "maximum". It has a probability distribution with an expectation value E and a variance V. It is important to realize that tuning θ in order to maximize $E[P(\theta)]$ is just one of the available options. Maximizing $E[P(\theta)]$ provides the highest mean performance. Another possibility is to maximize the lower limit of a confidence interval. Since the risk factor is always a major concern in investments, and since the spread between individual years obviously can be very high, this sounds like a promising idea. A lower limit P_{low} for a confidence interval could be defined as:

$$P_{low} = E[P(\theta)] - \sqrt{V[P(\theta)]}, \quad (16.16)$$

where $V[P(\theta)]$ is the variance of the stochastic variable $P(\theta)$. Yet another possibility is to use the Sharpe Ratio SR, which expresses the excess return in units of its standard deviation as:

$$SR = \frac{E[P(\theta)]}{\sqrt{V[P(\theta)]}}. \quad (16.17)$$

The Sharpe Ratio is normally used to evaluate the performance of a trading strategy (Sharpe [30, 31]). However, Choey and Weigend [8] propose to use the Sharpe Ratio as an objective function in portfolio optimization and derive a learning algorithm for artificial neural networks. The Sharpe Ratio should be as high as possible for an optimal trading algorithm.

Due to the high noise level in the data, we have added a modified Sharpe Ratio, where the outliers have been removed. The largest and smallest $P_i(\theta)$ in Expression (16.15) have been removed before taking the expected value and standard deviation for each θ. The unmodified Sharpe Ratio is shown in the top-right diagram, and the one with outliers removed in the mid-right diagram. As we can see, the one with the outliers removed clearly reveals a maximum at around 20-25, followed by a clear decline. The unmodified Sharpe Ratio shows no such pattern.

16.7 Other ASTA Features

By adding a PLOT parameter in the call to many of the predefined ASTA functions, illuminating graphs are generated. The call to the previously used Stoch function for example, may look like:

$$\text{Stoch(30,3,3,20,80,PLOT)}>0. \tag{16.18}$$

The enabled PLOT option generates separate graphs with the signals from the Stoch function. In Figure 16.6 the components of the Stochastics indicator are displayed. The two horizontal lines mark the buy level (80%) and the sell level (20%). When the "Oscillator %K" value passes these lines, a buy or sell signal is issued.

A problem with a trading-rule-based system like ASTA is that commonly used methods for modeling or inductive learning, such as regression models and artificial neural networks, cannot be applied in a normal fashion. The reason for this is that a set of training data with patterns and targets is not available in the same way as in the Time Series Approach. However, a related modeling problem can be formulated as an extension to a system based on trading rules. The points where the system signals buy or sell, can be used to extract examples that can be input to a modeling or classification routine. The trading rule may be refined in this way to produce a higher hit rate and most probably a higher overall profit. Therefore, a data extraction function has been built into the ASTA system. In Figure 16.5, the "Dump trades to window" box has been checked. All generated trades are dumped in ASCII form in the MATLAB command window (Figure 16.7). By "Dump trades to file", a data file for post-processing with classification tools is written to the disk.

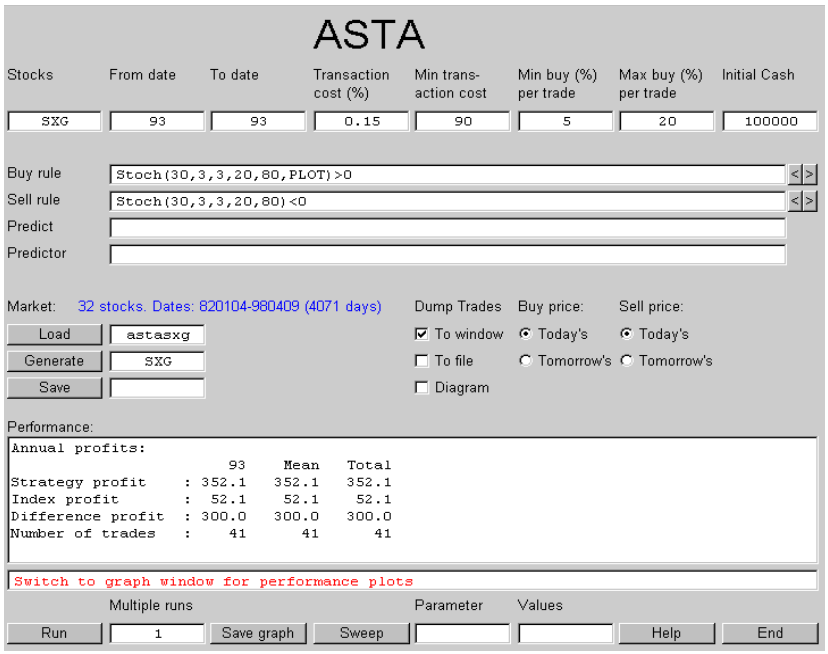

Figure 16.5: ASTA command window with Stochastic buy and sell rules. The PLOT option in the Buy rule generates separate plots with signals for each stock. The "Dump trades to window" check box generates a list of all trades in the MATLAB command window.

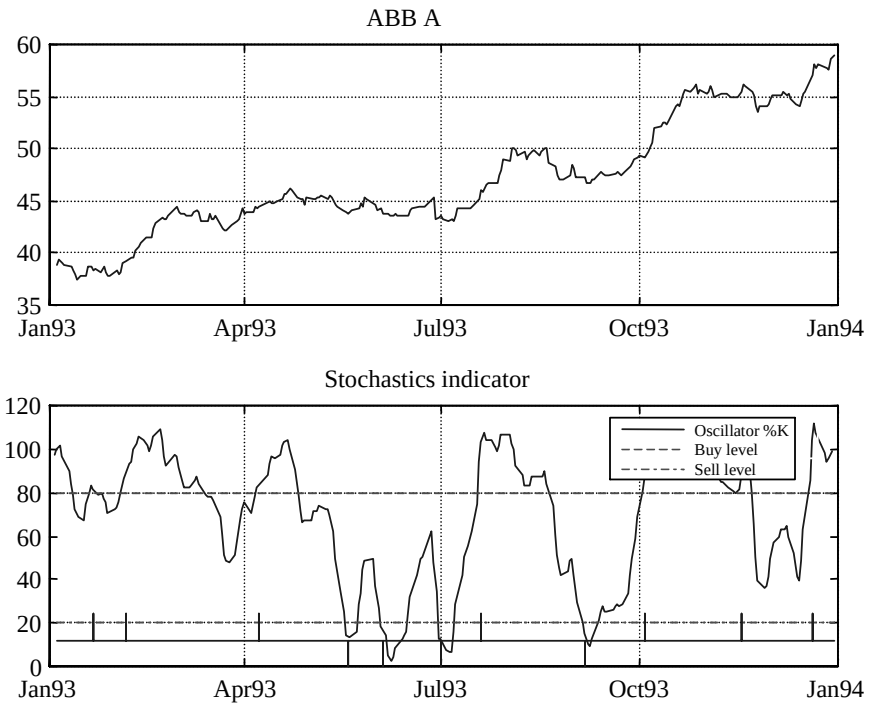

Figure 16.6: Stochastics buy and sell signals for the ABB stock. Upgoing bar denotes a buy signal and downgoing bar denotes a sell signal.

Trade	Date	No.	Stock	Price	Total Profit (%)		Wealth	Cash
1	930105	77	21 Skandia	85.27	6566.05			93343.95
2	930105	79	27 Sydkraft C	82.60	6525.22			86728.73
3	930105	1535	31 Allgon B	4.28	6574.72		99730.00	80064.01
4	930107	375	20 SHB A	26.34	9875.96			70119.50
5	930107	329	32 Nokia A	30.00	9870.00		99585.62	60159.50
6	930108	349	10 Hennes & Mauritz	28.20	9841.80			50260.08
7	930108	323	25 SSAB A	30.50	9851.50		99860.44	40318.58
8	930115	431	7 Avesta Sheffield	22.68	9776.63			30528.47
9	930115	351	20 SHB A	27.84	9771.77		100374.14	20666.71
10	930121	258	2 ABB A	38.40	9907.20			10702.32
11	930121	64	14 Kinnevik B	81.00	5184.00		NaN	5428.32
12	930127	54	21 Skandia	97.46	5262.68		NaN	84.48
13	930302	-64	14 Kinnevik B	79.00	-5056.00	-2.47	128010.13	5051.23
14	930322	-131	21 Skandia	93.71	-12275.62	3.78	131216.32	17260.75
15	930323	-431	7 Avesta Sheffield	23.16	-9980.37	2.08	NaN	27165.09
16	930325	-323	25 SSAB A	33.75	-10901.25	10.66	132663.45	37991.32
17	930329	4897	31 Allgon B	5.45	26675.18		132626.85	11247.39
18	930406	2031	31 Allgon B	5.51	11186.34		132246.33	1.77
19	930519	-258	2 ABB A	43.70	-11274.60	13.80	164709.29	11186.39
20	930525	113	22 Skanska B	97.50	11017.50		NaN	94.54
21	930608	-329	32 Nokia A	46.25	-15216.25	54.17	187009.69	15221.10
22	930610	154	27 Sydkraft C	97.95	15083.58		184210.74	54.44
23	930621	-349	10 Hennes & Mauritz	38.00	-13262.00	34.75		13226.55
24	930621	132	22 Skanska B	99.00	13068.00		185878.28	68.55
25	930716	-233	27 Sydkraft C	93.56	-21799.15	0.88	197758.60	21778.13
26	930719	308	7 Avesta Sheffield	34.97	10770.39		197540.39	10922.70
27	930720	235	2 ABB A	46.00	10810.00		199141.88	30.11
28	930906	-235	2 ABB A	47.20	-11092.00	2.61	273799.16	11032.41
29	930916	-308	7 Avesta Sheffield	36.86	-11352.70	5.41	272928.45	22322.39
30	930920	227	9 Ericsson B	97.83	22207.73		272118.39	33.78
31	930921	-245	22 Skanska B	138.00	-33810.00	40.37	270682.53	33753.80
32	930922	-726	20 SHB A	95.00	-68970.00	251.03	269365.98	102627.47
33	930923	672	32 Nokia A	80.00	53760.00		270719.82	48798.81
34	930927	562	10 Hennes & Mauritz	43.20	24278.40		282121.66	24450.66
35	931004	495	2 ABB A	49.20	24354.00		309655.25	42.43
36	931108	-227	9 Ericsson B	102.87	-23350.81	5.15	NaN	23303.54
37	931111	136	21 Skandia	170.55	23194.39		429195.43	42.48
38	931126	-672	32 Nokia A	100.50	-67536.00	25.62	399726.79	67477.30
39	931203	764	30 Volvo B	88.20	67384.19		387109.87	85.89
40	931215	-136	21 Skandia	159.30	-21664.94	-6.59	434649.03	21661.02
41	931221	127	22 Skanska B	169.00	21463.00		443866.23	132.74

Figure 16.7: Dump of generated trades from Stochastic trading rules for the year 1993.

16.8 Results and Further Development

The presented system provides a powerful tool for the development and evaluation of trading algorithms. Parameter settings can be tested and data screening can be performed easily interactively. As was mentioned in Section 16.1.1, one of the reasons of the development of the ASTA system, was to use it as an objective function when tuning model parameters with global optimization, and to find the general structure of trading rules, for example within a genetic framework. The preliminary results show both possibilities and difficulties. The track can be examined considerably.

The dangers of "data snooping" got highlighted by breaking down the performance measures into shorter intervals. However, the inherent uncertainty resulting from the noisy processes involved, calls for more computer-intensive simulation schemes in order to achieve statistically significant performance measures. Methods such as those described in Section 16.2.4 will be implemented and evaluated.

Well after all is said and done,
Gotta move, while it's still fun,
So let me walk before they make me run
Before They make Me Run, Jagger/Richards

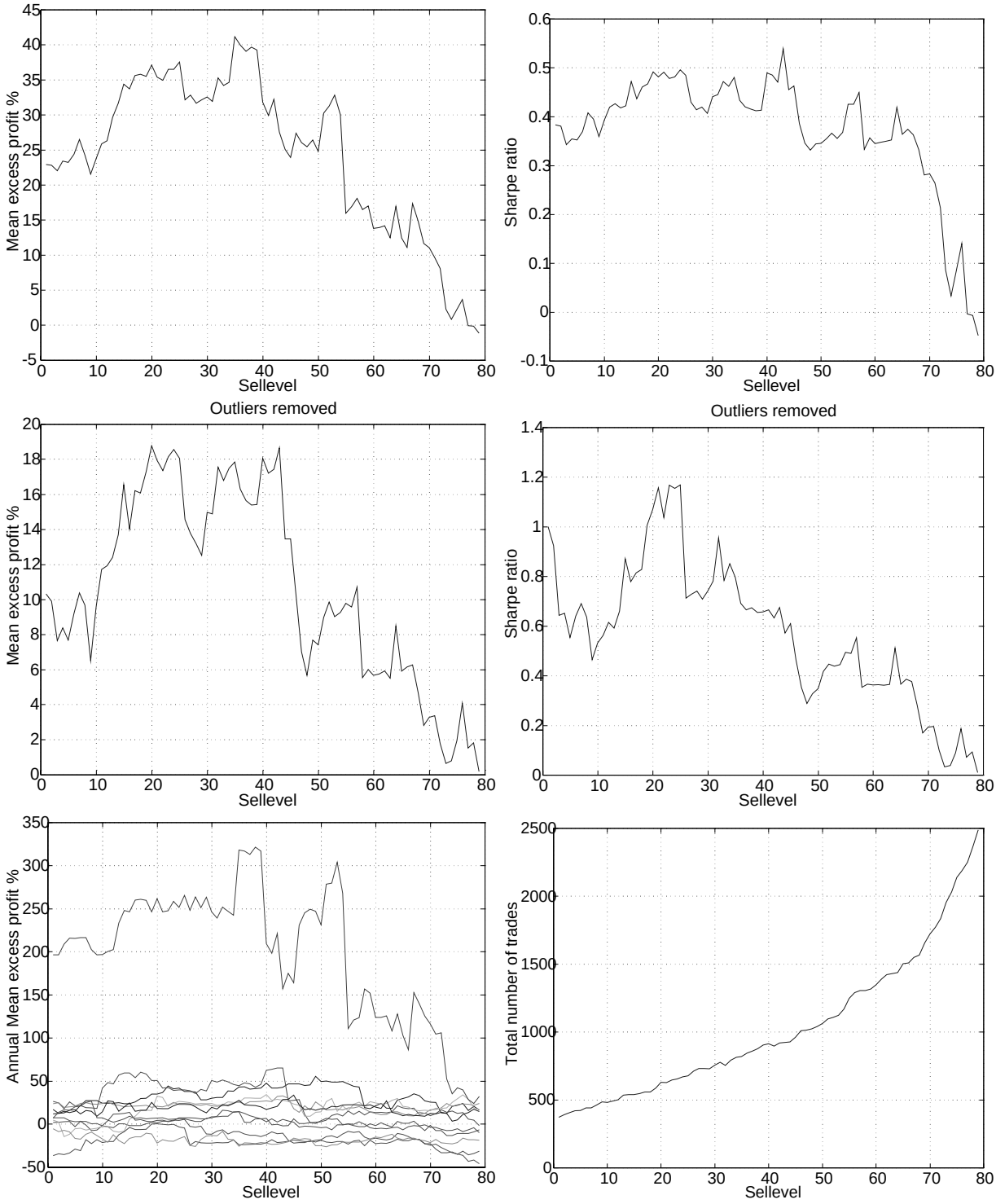

Figure 16.8: Trading results as a function of the Sellevel parameter. Stocks: SXG. Years:87-97. Buy rule: Stoch(30,3,3,Sellevel,80)>0. Sell rule: Stoch(30,3,3,Sellevel,80)<0

Bibliography

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

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

[3] A. Atiya. Design of time-variable stop losses and profit objectives using neural networks. In A. S. Weigend, Y. S. Abu-Mostafa, and A.-P. N. Refenes, editors, *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, pages 76–83, Singapore, 1997. World Scientific.

[4] 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.

[5] Y. Bengio. Training a neural network with a financial criterion rather than a prediction criterion. In A. S. Weigend, Y. S. Abu-Mostafa, and A.-P. N. Refenes, editors, *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, pages 36–48, Singapore, 1997. World Scientific.

[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. In A. S. Weigend and N. A. Gershenfeld, editors, *Time Series Prediction: Forecasting the Future and Understanding the Past*, pages 347–366, Reading, MA, 1994. Addison-Wesley.

[8] M. Choey and A. S. Weigend. Nonlinear trading models through Sharpe Ratio maximization. In A. S. Weigend, Y. S. Abu-Mostafa, and A.-P. N. Refenes, editors, *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, pages 3–22, Singapore, 1997. World Scientific.

[9] G. Cybenko. Approximation by superposition of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2:303–314, 1989.

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

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

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

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

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

[15] G.Leitch and J.Tanner. Economic forecast evaluation: Profit versus the conventional error measures. *The American Economic Review*, pages 580–590, 1991.

[16] G. Hawawini and D. B. Keim. On the predictability of common stock returns: World-wide 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.

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

[18] T. Hellström and K. Holmström. Predictable Patterns in Stock Returns. Research and Reports Opuscula ISRN HEV-BIB-OP–30-SE, Mälardalen University, Västerås, Sweden, 1998.

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

[20] B. LeBaron. Persistence of the dow jones index on rising volume. Technical report, Department of Economics, University of Wisconsin - Madison, Madison, WI, July 1992.

[21] 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.

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

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

[24] 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.

[25] J. E. Moody. Shooting craps in search of an optimal strategy for training connectionist pattern classifiers. In J. E. Moody, S. J. Hanson, and R. P. Lippmann, editors, *Advances in Neural Information Processing Systems 4, Proceedings of the 1991 NIPS Conference*, pages 847–854, San Mateo , CA, 1992. Morgan Kaufmann Publishers.

[26] J. E. Moody and L. Z. Wu. Optimization of trading systems and portfolios. In A. S. Weigend, Y. S. Abu-Mostafa, and A.-P. N. Refenes, editors, *Decision Technologies for Financial Engineering (Proceedings of the Fourth International Conference on Neural Networks in the Capital Markets, NNCM-96)*, pages 23–35, Singapore, 1997. World Scientific.

[27] D. Ormoneit and R. Neuneier. Reliable neural network predictions in the presence of outliers and Non-Constant variences. 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 578–587, Singapore, 1996. World Scientific.

[28] 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.

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

[30] W. F. Sharpe. Mutual fund performance. *Journal of Buisness*, pages 119–138, January 1966.

[31] W. F. Sharpe. The Sharpe Ratio. *Journal of Portfolio Management*, 21:49–58, 1994.

[32] 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.

[33] 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.

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

[35] H. Tong and K. S. Lim. Threshold autoregression, limit cycles and cyclical data. *J. Roy. Stat. Soc. B*, 42:245–292, 1980.

[36] R. R. Trippi and J. K. Lee. *Artificial Intelligence in Finance and Investing*. Irwin Professional Publishing, Chicago, 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, B. A. Huberman, and D. E. Rumelhart. Predicting sunspots and exchange rates with connectionist networks. In M. Casdagli and S. Eubank, editors, *Nonlinear Modeling and Forecasting, SFI Studies in the Sciences of Complexity, Proc. Vol XII*, pages 395–432. Addison-Wesley, 1992.

[40] A. S. Weigend and B. LeBaron. Evaluating neural network predictors by bootstrapping. In *Proceedings of International Conference on Neural Information Processing (ICONIP'94)*, pages 1207–1212, 1994. Technical Report CU-CS-725-94, Computer Science Department, University of Colorado at Boulder.

[41] 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.

[42] 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.

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

Index