NTNU
Norwegian University of
Science and Technology

Predicting Stock Prices Using Technical Analysis and Machine Learning

Jan Ivar Larsen

Master of Science in Computer Science
Submission date: June 2010
Supervisor: Helge Langseth, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

Problem Description

In this thesis, a stock price prediction model will be created using concepts and techniques in technical analysis and machine learning. The resulting prediction model should be employed as an artificial trader that can be used to select stocks to trade on any given stock exchange. The performance of the model will be evaluated on stocks listed on the Oslo Stock Exchange.

Assignment given: 15. January 2010
Supervisor: Helge Langseth, IDI

Abstract

Historical stock prices are used to predict the direction of future stock prices. The developed stock price prediction model uses a novel two-layer reasoning approach that employs domain knowledge from technical analysis in the first layer of reasoning to guide a second layer of reasoning based on machine learning. The model is supplemented by a money management strategy that use the historical success of predictions made by the model to determine the amount of capital to invest on future predictions. Based on a number of portfolio simulations with trade signals generated by the model, we conclude that the prediction model successfully outperforms the Oslo Benchmark Index (OSEBX).

Preface

This report constitutes my master thesis, written and implemented during the 10th semester of the Master of Science studies in Computer Science at the Norwegian University of Science and Technology (NTNU). The work was initiated and executed at the Department of Computer and Information Science (IDI), Division of Intelligent Systems (DIS), starting on 15th of January 2010 and ending on 15th of June 2010.

The problem description was conceptualized in cooperation with my professor and supervisor Helge Langseth. I am grateful and humbled by the time and effort Langseth has devoted to supervising the project, and his enthusiasm for offering advice, support and feedback has been an inspiration and significantly increased the quality and extent of the work.

Trondheim, June 15, 2010.

Jan Ivar Larsen

Contents

1 Introduction

Forecasting the direction of future stock prices is a widely studied topic in many fields including trading, finance, statistics and computer science. The motivation for which is naturally to predict the direction of future prices such that stocks can be bought and sold at profitable positions. Professional traders typically use fundamental and/or technical analysis to analyze stocks and make investment decisions. Fundamental analysis is the traditional approach involving a study of company fundamentals such as revenues and expenses, market position, annual growth rates, and so on (Murphy, 1999). Technical analysis, on the other hand, is solely based on the study of historical price fluctuations. Practitioners of technical analysis study price charts for price patterns and use price data in different calculations to forecast future price movements (Turner, 2007). The technical analysis paradigm is thus that there is an inherent correlation between price and company that can be used to determine when to enter and exit the market .

In finance, statistics and computer science, most traditional models of stock price prediction use statistical models and/or neural network models derived from price data (Park and Irwin, 2007). Moreover, the dominant strategy in computer science seems to be using evolutionary algorithms, neural networks, or a combination of the two (evolving neural networks). The approach taken in this thesis differ from the traditional approach in that we use a knowledge-intensive first layer of reasoning based on technical analysis before applying a second layer of reasoning based on machine learning. The first layer of reasoning thus performs a coarse-grained analysis of the price data that is subsequently forwarded to the second layer of reasoning for further analysis. We hypothesis that this knowledge-intensive coarse-grained analysis will aid the reasoning process in the second layer as the second layer can then focus on the quintessentially important aspects of the price data rather than the raw price data itself.

1.1 Purpose

The purpose of the thesis is to create a stock price prediction model for the Oslo Stock Exchange. The resulting model is intended to be used as a decision support tool or as an autonomous artificial trader if extended with an interface to the stock exchange. A high-level system overview of the developed stock price prediction model is presented in Figure 1.1.

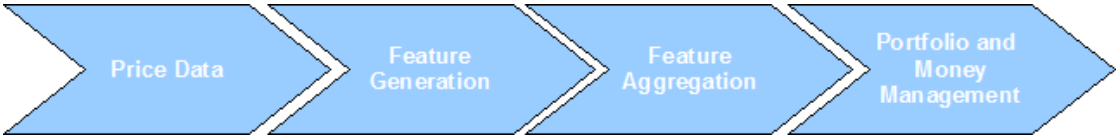


Figure 1.1: The Stock Price Prediction Model

The developed model employs a two-layer reasoning approach. The first reasoning layer is a knowledge-intensive *Feature Generation* module based on domain knowledge from technical analysis and certain other statistical tools. This is implemented by a set of artificial agents where each agent employs a specific subset of expertise from technical analysis. The resulting output is a set of quintessentially important feature-values derived from the price data (such as price is trending up, a trend reversal is expected to occur, the stock is trading on high volume, etc.). The generated feature-values are then forwarded to the second layer of reasoning called the *Feature Aggregation* module. In the Feature Aggregation module machine learning is employed to learn a classification model that aggregate and place the derived feature-values in context of each other. The resulting output is an investment strategy that can be used to select stocks to trade on the Oslo Stock Exchange. In the *Portfolio and Money Management* module the performance of the investment strategy is evaluated in terms of profitability by simulating portfolio runs using trade signals generated by the investment strategy. Moreover, the module includes a money management strategy used to assess the *strength* (i.e., confidence) of generated predictions and to determine the amount of capital to invest on a generated trade signal.

1.2 Scope

Our goal with the Feature Generation module is to provide a knowledge-intensive and computationally efficient coarse-grained analysis of *historical prices* which can be analyzed further in a second layer of reasoning. The domain knowledge implemented in the module is thus limited to methods and techniques in technical analysis. The technical analysis literature includes a wealth of different stock analysis techniques, some of which involve complicated and intricate price patterns subjective in both detection and interpretation. These methods would be both computationally expensive to detect and evaluate, and have consequently been disregarded. We thus apply Occam's razor to the choice of methods in technical analysis, focusing on the most popular indicators that can be efficiently operationalized and are intuitive in interpretation.

It may seem overly presumptuous to believe that historical price fluctuations alone can be used to predict the direction of future prices. It may thus seem natural to include some fundamental analysis knowledge in the feature generation process. However, due to the inherent limitations in time and the added complexity of including a second analysis technique, this has not been a priority. We have instead placed focus on creating a model that can be easily extended with new analysis techniques, not necessarily from technical analysis, by using two separate reasoning layers and using an agent-oriented approach for the domain knowledge. The agent-oriented approach is explained in detail in Chapter 3. Although our goal in this thesis is not to justify, prove or disprove technical analysis, by focusing strictly on technical indicators we are presented with an opportunity to evaluate the utility of selected methods in this form of stock analysis.

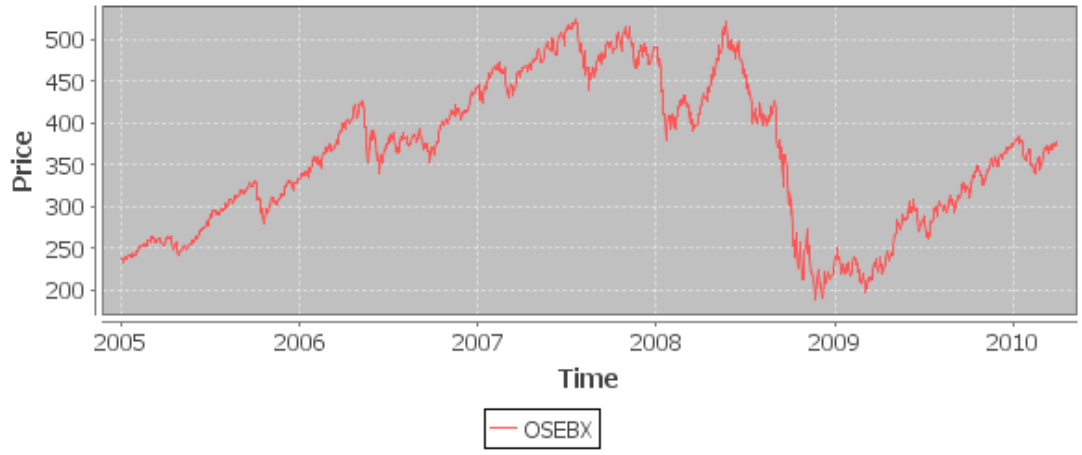


Figure 1.2: The Oslo Benchmark Index from 01-01-2005 to 01-04-2010. Notice the sharp drop from mid-2008 to early 2009 resulting from the financial crisis.

1.3 The Dataset

The dataset available to learn and evaluate the performance of the implemented system includes daily historical prices available for the stocks currently listed on the Oslo Benchmark Index (OSEBX). The Oslo Benchmark Index is a weighted index of a representative selection of the stocks listed on the Oslo Stock Exchange. All stocks listed on the index are easily transferable which makes our model easier to validate as we can assume that selected stocks can be bought and sold at any time. The stocks used are listed in Appendix A along with a Python script that can be used to download the data from `www.netfonds.no`. The data used is all data available from 01-01-2005 to 18-05-2010. The data is always separated in a training and test set where the separation point is configurable and will be noted in the report when needed. Although our focus in this thesis is on stocks listed on the Oslo Stock Exchange, the developed model can just as easily be used for any other stock exchange where a sufficient amount of daily historical prices are available.

1.4 Success Criteria

The purpose of the thesis is to create a stock price prediction model that can be used as a decision support tool or as an autonomous artificial trader. The central research question thus becomes, *"can we create a computer system that trades in stocks with performance comparative to a professional trader"*? The primary measure of success will thus be based on executing portfolio runs that simulate transactions based on some initial investment capital (e.g., 100 000 NOK) and an investment strategy generated by the model. The profits generated by portfolio simulations on the model will be evaluated by comparing it against investing the entire initial investment capital in the Oslo Benchmark Index

shown in Figure 1.2. If it is more profitable to use the developed stock price prediction model to select stocks to trade, we consider the model a success.

1.5 Document Structure

The remainder of this thesis document is organized in the following six chapters,

Background and Rationale contains a non-technical overview of trading basics and traditional stock analysis techniques including the rationale behind technical analysis.

Feature Generation documents different methods in technical analysis and the first layer of reasoning including the agent population designed to execute the feature generation process.

Feature Aggregation describes the second layer of reasoning based on machine learning, evolutionary algorithms and decision tree classification.

Portfolio and Money Management describes the money management strategy and the portfolio simulation procedure.

Results and Discussion documents the results obtained by testing the developed model on stocks listed on the Oslo Stock Exchange.

Conclusion contains concluding remarks and points for future work.

2 Background and Rationale

In this chapter the basics of stock markets, trading, and general price prediction techniques are introduced. The primary focus will be on the fundamentals that govern stock markets and the chosen stock analysis technique; technical analysis. Our goal here is not to go into specific details of methods in technical analysis, but rather give an overview of the underlying rationale in the field. Methods in technical analysis that have been implemented in the prediction model will be elaborated in Chapter 3. The chapter is concluded with an overview of related research on operationalized technical analysis. Please note that most of the material presented in this chapter was initially researched and written in a preliminary project (Larsen, 2009), and is presented here in succinct form for completeness and for readers unfamiliar with the preliminary project.

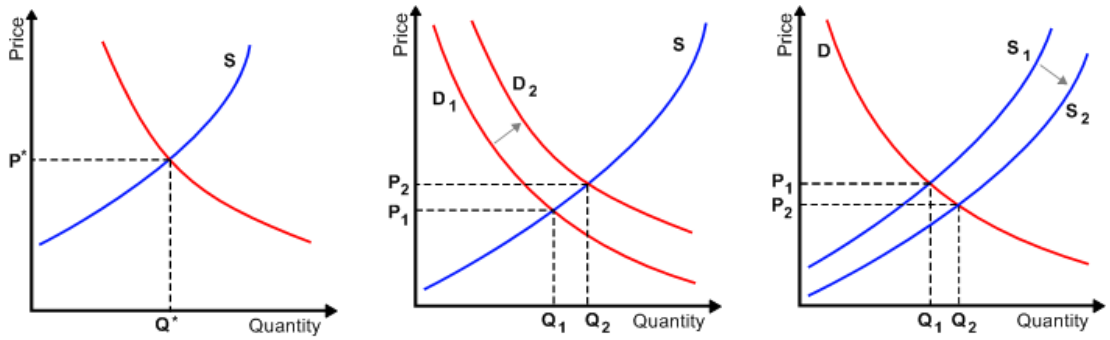
2.1 Trading Basics

Trading stocks is the process of buying and selling shares of a company on a stock exchange with the aim of generating profitable returns. The stock exchange operates like any other economic market; when a buyer wants to buy some quantity of a particular stock at a certain price, there needs to be a seller willing to sell the stock at the offered price. Transactions in the stock market are processed by brokers who mediate sales between buyers and sellers. Brokers typically charge a commission fee for completed transactions (e.g., a fixed amount for each transaction or a small percentage of the order total). Naturally, buyers want to minimize the price paid for the stock and sellers want to maximize the selling price for the stock. The stock market is thus governed by the same fundamental economic principles as any other economic market, namely supply and demand.

2.1.1 Supply and Demand

Supply and demand is one of the most fundamental concepts in economic theory and the backbone of economic and fundamental forecasting (Murphy, 1999). The supply and demand curve given in Figure 2.1 show the relationship between supply (provided by the sellers) and demand (provided by the buyers). At price equilibrium (Figure 2.1a), when the supply curve intersects with the demand curve, the seller and buyer agree on a price and a transaction can occur. In our case, this would involve a buyer of some quantity of shares Q^* of a stock at price P^* as provided by some seller.

In Figure 2.1b we see a right shift of the demand curve meaning that, for some reason, demand has increased. This increase in demand (i.e., there is an increase in the number of buyers resulting in a situation with more buyers than sellers), creates an increase in price from p_1 to p_2 which becomes the new price equilibrium. Traders essentially want



(a) Price equilibrium at price P^* and quantity c.

(b) An increase in demand causes an increase in price.

(c) An increase in supply causes a reduction in price.

Figure 2.1: The relationship between supply and demand, figure adapted from `www.wikipedia.com`

to recognize this shift in demand before it happens so that the stock can be purchased at a price close to p_1 and sold at a price close to p_2, making a profit of $p_2 - p_1$. Similarly, Figure 2.1c shows an increase in supply (i.e., there are more sellers than buyers) that results in a decrease in price, which we would like to establish early so that we can a) buy the stock at p_2, b) refrain from buying the stock at p_1, or c) *short sell* the stock at p_1.

2.1.2 Short Selling

The process of short selling a stock is done to profit from a price decline and involves selling a stock with the intention of buying it back later at a lower price (Turner, 2007). Essentially, this involves a broker lending the trader a fixed number of shares of a stock which are then sold and the profits are credited the trader. Eventually, the trader will have to *cover* the position by returning the same number of shares to the broker, profiting if the stock can be repurchased at a lower price. For example, 100 shares of some stock are sold short at 5 NOK and the trader is credited with 500 NOK. If the same number of shares can be repurchased later at 3 NOK, the loan can be returned with a cost of 300 NOK, making a profit of 200 NOK.

2.1.3 Trading Time Frames

Turner (2007) describes four basic trading time frames that are commonly used by traders:

Position trades: stocks may be held from weeks to months.

Swing trades: stocks may be held for two to five days.

Day trades: stocks are bought and sold within the same day.

Momentum trades: stocks are bought and sold within seconds, minutes or hours.

Each time frame has its own risk-reward ratio where shorter time frames are typically associated with greater risk (Turner, 2007). Due to the computational capacity of a

computer system we would ideally like to focus on the two shorter time frames, day and momentum trades (hypothesizing that a computer system would give us an edge over the other market players as it would have the ability to harvest and analyze information much quicker than a human trader). However, due to the availability of data sets consisting of daily stock prices, our primary focus in this thesis will be on *swing trades*. The techniques used and the prediction model implemented will be built to be extended with little effort to support day trading and momentum trades.

2.1.4 Charting Techniques

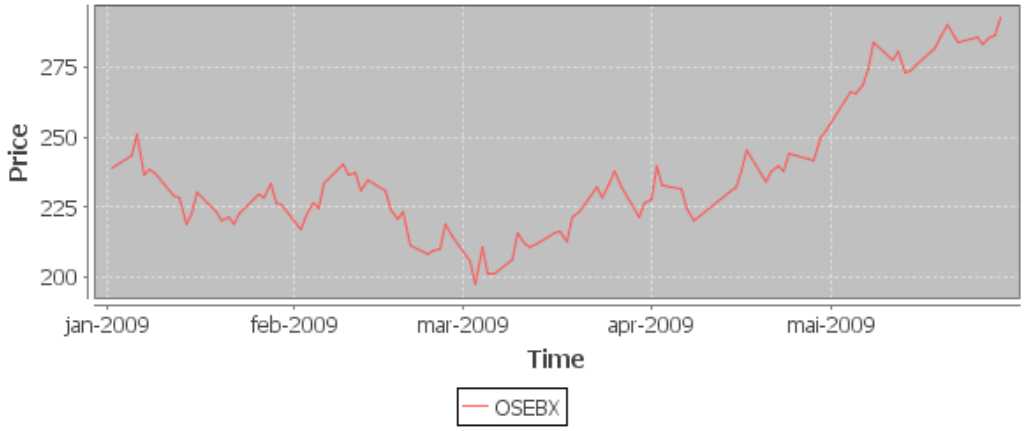
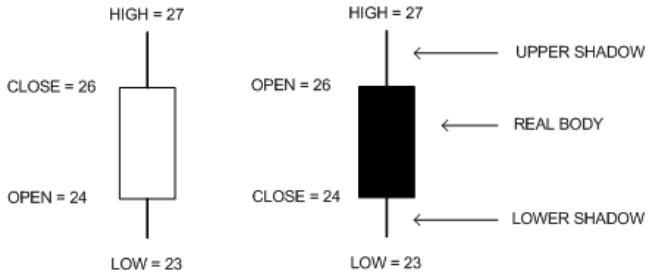


Figure 2.2: Closing Price Plot

A price chart shows how a stock's price has evolved over a given period of time, typically presented as in Figure 2.2 which plots the closing price of the Oslo Benchmark Index (OSEBX) over the beginning of 2009. Closing price plots provide a good perspective on the trending direction of the price, but hides some potentially useful information. Within any time frame, stock prices reach four different levels, namely open, close, high and low. The opening price is the price at which the stock was traded for at the start of the period, and similarly the closing price is the price at the end of the period. The high and low price refers to the highest and lowest price obtained for the stock during the period. The price levels thus reveal more about how the stock has been traded during the period, and consequently provides a greater basis for analyzing where the stock will go in the next period.

Candlestick charts are designed to give the trader a quicker and more complete picture of price movements in a given time frame. The candlestick entity (named so for its resemblance to a candle with a wick in both ends), as shown in Figure 2.3, is drawn with a rectangular "real body" that represents the range between the opening and closing price, and lower and upper "shadows" that represent the lowest and highest price obtained for the stock in the period. The real body is colored white or green if the closing price is above the opening price, and black or red if the closing price is below the opening price. Thus, green candlesticks represent positive price movements and red candlesticks represent negative price movements. In this way, the candlestick charts provides a more visual and immediate impression of price movements. Candlestick charts can be used on



(a) *close* > *open* (b) *close* < *open*

Figure 2.3: Candlestick entity, figure adapted from Turner (2007)

any time scale, from intraday 15-minute charts to daily or yearly charts. According to preference and scope of analysis, traders use different time scales when analyzing a stock. Figure 2.4 shows a daily candlestick chart over the same period as in Figure 2.2.

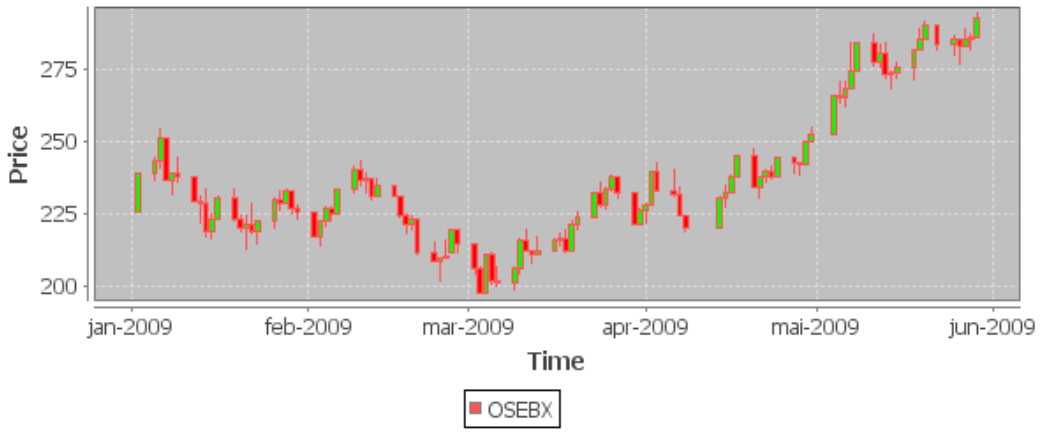


Figure 2.4: Candlestick Plot

2.2 Technical Analysis

Investors and traders typically employ two classes of tools to decide what stocks to buy and sell; fundamental and technical analysis, both of which aim at analyzing and predicting shifts in supply and demand (Turner, 2007). As mentioned earlier, shifts in supply and demand is the basis of most economic and fundamental forecasting. If there are more sellers than buyers for a stock (i.e., increased supply), the theory states that the price should fall, and similarly, if there are more buyers than sellers (i.e., increased demand) the price should rise. Given the ability to foresee these shifts in supply and demand thus gives the trader the ability to establish profitable entry and exit positions, which is the ultimate goal of stock analysis.

While fundamental analysis involves the study of company fundamentals such as revenues and expenses, market position, annual growth rates, and so on, technical analysis is solely

concerned with price and volume data, particularly price patterns and volume spikes (Turner, 2007). Price and volume data is readily available in real time, which makes technical analysis ideally suited for short-term swing trades. The underlying assumption in technical analysis is that stock prices evolve with a certain regularity, forming reliable and predictable price and volume patterns that reveal market psychology which can be used to determine shifts in supply and demand (Turner, 2007). This assumption might seem overly presumptuous, and as a result, the next few sections will be devoted to the premise and psychological rational on which the technical approach to stock analysis is based.

2.2.1 The Three Premises on which Technical Analysis is Based

Murphy (1999) describes three premises on which technical analysis is based:

1. Market action discounts everything.
2. Prices move in trends.
3. History repeats itself.

Market action discounts everything

Market action is defined by Murphy (1999) as the sources of information available to the trader (i.e., price and volume data). By assuming that market action discount everything we are essentially assuming that everything that could influence the price (that is, fundamentals, politics, psychology, etc.) is integrated and reflected in the price and volume data. Price thus indirectly provides a perspective of the fundamentals and a study of price action is therefore all that is required to predict shifts in supply and demand. For example, if prices are rising, the technician assumes that, for whatever specific reason, demand must exceed supply and the fundamentals must be positive. Practitioners of technical analysis thus believe that there is an inherent correlation between market action and coompany that can be used to forecast the direction of future prices.

Prices move in trends

A price trend is the prevailing direction of a stock's price over some period of time. The concept of trend is perhaps *the* quintessential idea in technical analysis and as we'll see in Chapter 3 most technical indicators are designed to identify and follow existing trends (Turner, 2007). What we are basically looking for when doing technical analysis is patterns in the price data that signal continuations or reversals in trend. We want to recognize situations that signal a continuation in trend so that we can "ride" the trend as long as possible. We also want to look for situations that signal a reversal in trend so we can a) sell the stock before the trend turns, or b) buy the stock at the moment it reverses. For example, if we hold a particular stock in an uptrend, we look for continuations in the uptrend to confirm our position, and reversals so that we can exit the position before the stock goes into a downtrend, thereby maximizing potential profits. When analyzing and picking stocks we thus look for stocks that are trending, try to analyze the strength

of the trend, and either buy or sell depending on our current position. Thus, for the methods in technical analysis to have any value, we have to assume that prices does form in trends.

History repeats itself

When trading with technical analysis we examine stock price data for price patterns that in some way predict the direction of price in the future. We consequently have to assume that price patterns form with a certain regularity and that price patterns that have been successful in the past will be successful in the future. As financial markets are fueled by human actions and expectations, Murphy (1999) attributes the formation of regular and predictive price patterns and price calculations to a study in human psychology and group dynamics which is the basis for *behavioral finance*.

2.2.2 Behavioral Finance

Early financial theory was predominantly based on the efficient markets hypothesis (EMH). The efficient markets hypothesis was originally stated in (Fama, 1965) and says that the price of traded assets (e.g., stocks) are informationally efficient, meaning prices always fully reflect all known information and instantly change to new information, and all agents in the market are utility maximizing and have rational expectations. Given this assumption, any attempt at analyzing past price and trading stocks would be a waste of time as it would be impossible to consistently outperform the market because all known information is integrated in the price and all agents value the information equally (Fama, 1965; Shleifer, 2000). The theory was supported by successful theoretical and empirical work, and was widely considered to be proved. However, from its height of dominance around the 1970s to the 1990s, it has been challenged by and focus has shifted towards behavioral finance (Shiller, 2003).

Behavioral finance looks at finance from a broader social science perspective, including theory from psychology and sociology. Human desires, goals, motivations, errors and overconfidence are thus included as factors that affect finance (Shefrin, 2002). It follows hence that investors cannot be viewed as utility maximizing agents with rational expectations. Rather, when two investors are confronted with the same price information, their reactions will be different, and they will value the information in different ways. As pointed out by Turner (2007), when a trader buys a stock at a certain price p it is certainly with expectations that it will rise. In much the same way, the seller at price p is probably expecting the price to drop. Only one of them can win and make a profit. This difference in valuation is what drives market changes, trends, and profitable situations. Turner (2007) thus classifies *greed* and *fear* as primary emotions that drive the market.

2.2.3 Greed and Fear

In (Linløkken and Frölich, 2004; Turner, 2007) the motivation for technical analysis is largely based on human emotions, such as greed and fear, as a primary propellant of stock prices. Greed and fear are fundamental emotions that motivate nearly all traders.

For example, Turner (2007) describes the following situation as an example where greed takes hold of a trader, *"Say you buy a stock at the perfect entry point. Within minutes, the price rises even higher than you expected. A little voice whispers in your ear, This baby's gonna fly. Why don't you take all the money in your trading account and buy as many shares as you can? You'll make a killing!"* (Turner, 2007). Fear, on the other hand, might lead traders to sell shares prematurely. Irrational behavior in the market thus leads to profitable situations if detected. Price patterns are designed to determine this irrationality, when greed and fear takes hold of a stock. Turner (2007) explains that any successful trader need to learn how to control these emotions and rather rely on detecting when they occur in the market. In our case, a potential artificial trader is luckily endowed with a sense of bit-calculating rationality, so managing greed and fear should be a reasonable task.

2.2.4 Critique

Technical analysis is often criticized by academics for its lack of scientific and statistical validation (Murphy, 1999). In response, technical analysts often argue that technical analysis is a pragmatic discipline, largely interested in what works rather than existing theory. The fact remains, though, that a number of methods in technical analysis are highly subjective in nature, and critics often claim that price patterns and indicators used by practitioners of technical analysis is more in the mind and eye of the beholder. Nevertheless, practitioners vividly portray the utility of technical analysis, and its popularity has grown significantly during the past 10 years. This is most notably seen by the fact that most major newspapers are now posting stock advice based on technical analysis, and some brokerage firms (e.g., Christiania Securities in Norway) specialize in the use of technical analysis. Furthermore, research on the profitability of technical analysis has increased in volume and statistical significant during the past years. Park and Irwin (2007) recently did a review of research papers that try to analyze potential profits generated by technical analysis. They find that modern studies indicate that technical analysis consistently generate profitable returns in a variety of speculative markets (e.g., the stock market, foreign exchange market, et cetera).

In our case, technical analysis seems like an ideal approach to automate with a computer because stock prices are readily available. Furthermore, as many of the indicators in technical analysis are uncertain and difficult to interpret with purely analytical methods, it seems like a field ideally suited for artificial intelligence and machine learning. However, although we will limit our study in this thesis to methods in technical analysis, other stock analysis techniques will be included as an important point for future work.

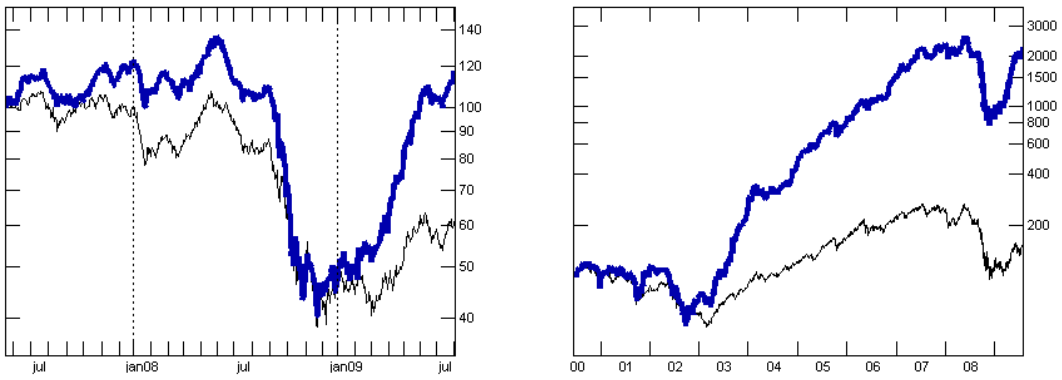
2.3 Related Research

In this section several systems and research papers that have investigated the profitability of computerized technical analysis are presented. Throughout the report we will continue to investigate the predictability of the techniques discussed. This section thus serves as an overall, high-level, introduction to this work.

Brock et al. (1992) describe an approach that employs two technical indicators (one of which is the moving average crossover rule discussed in detail in Section 3.2) to generate buy and sell signals. Profits generated by the signals are calculated by simulating transactions on the Dow Jones Industrial Average over 1897-1986. Their results shows that the system generate consistent returns. Specifically, investing on the buy signals produced by the technical indicator generate annual returns of 12%. Another interesting system is proposed by (Leigh et al., 2002). They propose a system that uses several techniques in artificial intelligence (e.g., neural networks and genetic algorithms) and several tools from technical analysis to analyze the stock exchange. They report positive result and excess returns compared to a simple buy-and-hold strategy.

Investtech[1] is a Norwegian company that sell subscriptions to a service that provides traders with a whole range of technical analysis tools for stocks on the Oslo Stock Exchange. The technical analysis they provide is primarily generated by automatic computer tools, although they also provide analyses that are supplemented with an evaluation from an expert technical analyst. As such, the services provided by Investtech are quite similar to the system that will be suggested in this report. However, we aim to take it one step further and allow the artificial trader to issue buy and sell orders on its own accord.

One of the most successful services provided by Investtech has been *today's case*. Each day, they publish an analysis of a stock that they consider technically positive. The recommendation includes an analysis by an expert, an automatically generated chart with technical indicators, an explanation of why the stock is recommended, and conclusively a buy or sell recommendation in the short and medium long run.



(a) Simulated today's case portfolio (blue curve) compared with Benchmark Index from May 2007 to July 2009.

(b) Simulated today's case portfolio (blue curve) compared with Benchmark Index from June 2000 to July 2009.

Figure 2.5: Results from simulated portfolio of today's case recommendations from Invesstech.

Investtech recently published a report documenting the profits generated by a simulated portfolio based on the today's case recommendations[2]. They show through a simulated portfolio of at most six stocks derived from recommendations posted as today's case

[1] `http://www.investtech.com/`

[2] `http://www.investtech.no/main/market.php?CountryID=1&p=info&fn=rptCaseStat0908`

that the recommendations yield considerable profits. Different scenarios with different time frames of investment are simulated. Their results show that the simulated portfolio rises more when the Oslo Benchmark Index (OSEBX) rises, and falls more when the benchmark index falls. This is an expected result as trading is generally considered riskier than placing all investments evenly over the stocks in the exchange. Their report does not mention how much the simulated portfolio fell compared to the benchmark index, but as we see from Figure 2.5a their portfolio follows the benchmark quite closely. In 2009, the simulated portfolio generated profits between 107% and 140% after 0.2% commission costs. These are considerable numbers considering that the benchmark index increased by only 32%.

3 Feature Generation

> *A good theoretical model of a complex system should be like a good caricature: it should emphasize those features which are most important and should downplay the inessential details. Now the only snag with this advice is that one does not really know which are the inessential details until one has understood the phenomena under study.* (Fisher, 1983)

The *Feature Generation* module implements the first layer of reasoning in the prediction model outlined in Section 1.1. The module implements a knowledge-intensive process that generates a set of discrete feature-values from the price data using domain knowledge from technical stock analysis. The generated features represent aspects of the data that are quintessential to stock price prediction. The process is essentially executing a coarse-grained analysis of the price data, filtering out the seemingly unimportant details and forwarding the seemingly important details in relation to stock price prediction.

Although the primary purpose of the module is to provide input to the next reasoning layer, the module may also serve as an independent prediction model. In this way, the domain knowledge implemented in the module can be tested and evaluated independently from other parts of the system. Moreover, the module should facilitate easy extension so that new analysis techniques, not necessarily from technical analysis, can be easily integrated in the module. As a result, the module was implemented using an agent-oriented approach where each agent is designed as an independent entity that implements a subset of knowledge from technical analysis. A conceptual illustration of the agent-oriented approach is given in Figure 3.1.

The technical analysis field includes a wide range of different methods and techniques. Some methods in technical analysis are based on intricate and complex price patterns that would be both computationally expensive to detect and subjective in interpretation. As we want the Feature Generation process to perform a coarse-grained analysis of the price data that is subsequently forwarded to a second layer of reasoning we also want the process to be relatively computationally efficient. We consequently apply Occam's razor to the choice of technical indicators, leaving out the intricate price patterns and focusing on the most popular indicators that can be efficiently operationalized and that are relatively objective in interpretation.

In order to facilitate easy extension of the agent population we define a formal agent interface that will be employed by every implemented agent,

$$\text{Agent}(P_s, t) \rightarrow \{v_1, v_2, \ldots\}$$

where P_s is the price history for some stock s and $\{v_1, v_2, \ldots\}$ is a set of discrete feature-values generated by the agent. For any time index t, $\text{Agent}(P_s, t)$ generates a feature-value

$v \in \{v_1, v_2, \ldots\}$ as a prediction for time $t + 1$. Thus, each agent in the agent population represents a feature that generates a fixed set of feature-values. The terms agent and feature will be used interchangeably in the remainder of the document. In cases where the domain knowledge implemented by an agent require additional parameters default values are always given so that the interface corresponds to the one above. An overview of the generated features is given in Table 3.1.

Some of the agents presented in this chapter was originally implemented during a previous project (Larsen, 2009). However, the agent population has been expanded and many of the old agents have been redesigned. A description of the entire agent population will consequently be included here for completeness.

Feature/Agent	Generated Values
Trend	Uptrend, Downtrend, Notrend
Moving Average Crossover	Buy, Sell, Hold
Candlestick	Buy, Sell, Hold
Stochastic	Buy, Sell, Hold
Volume	Strong-Volume, Weak-Volume
ADX	Strong-Trend, Weak-Trend

Table 3.1: Generated Features/Values

3.1 Trend Agent

The concept of price trends is perhaps *the* quintessential idea in technical analysis and trading in general (Murphy, 1999; Turner, 2007). The ability to identify price trends is primarily important for two reasons. For one, traders want to trade with the trend and "ride" the trend as long as possible (i.e., buying when price trends up and shorting when price trends down). Secondly, traders want to predict trend reversals (i.e., situations when the trend turns from uptrend to downtrend, or downtrend to uptrend) in order to establish good entry and exit positions, in which case being able to establish the preceding trend is essential.

Formally, a trend in any given time series $\mathbf{X} = (x_1, \ldots, x_n)$ is a prolonged period of time where $x_i \in \mathbf{X}$ rise or fall faster than their historical average (i.e., the prevailing direction of the data during some interval). A general approach used to identify the trending direction of a time series $\mathbf{X}$ is to first create a smoothed representation, $\mathbf{G}$, that roughly describes $\mathbf{X}$. Next, the first difference of $\mathbf{G}$ is calculated, $\mathbf{F}(t) = \mathbf{G}(t) - \mathbf{G}(t-1)$, and a trend is said to be established in intervals where the sign of the first difference is constant.

Figure 3.2 shows a plot of the Oslo Benchmark Index (OSEBX) with manually drawn trend lines (this plot will serve as a semi-benchmark when evaluating different trend identifying mechanisms). The plot highlights an important challenge that needs to be considered when selecting a trend identifying mechanism. Although the price over the entire interval is, in general, trending up, there are also several short-term downtrends in the price data. These short-term downtrends are important to detect as we would ideally

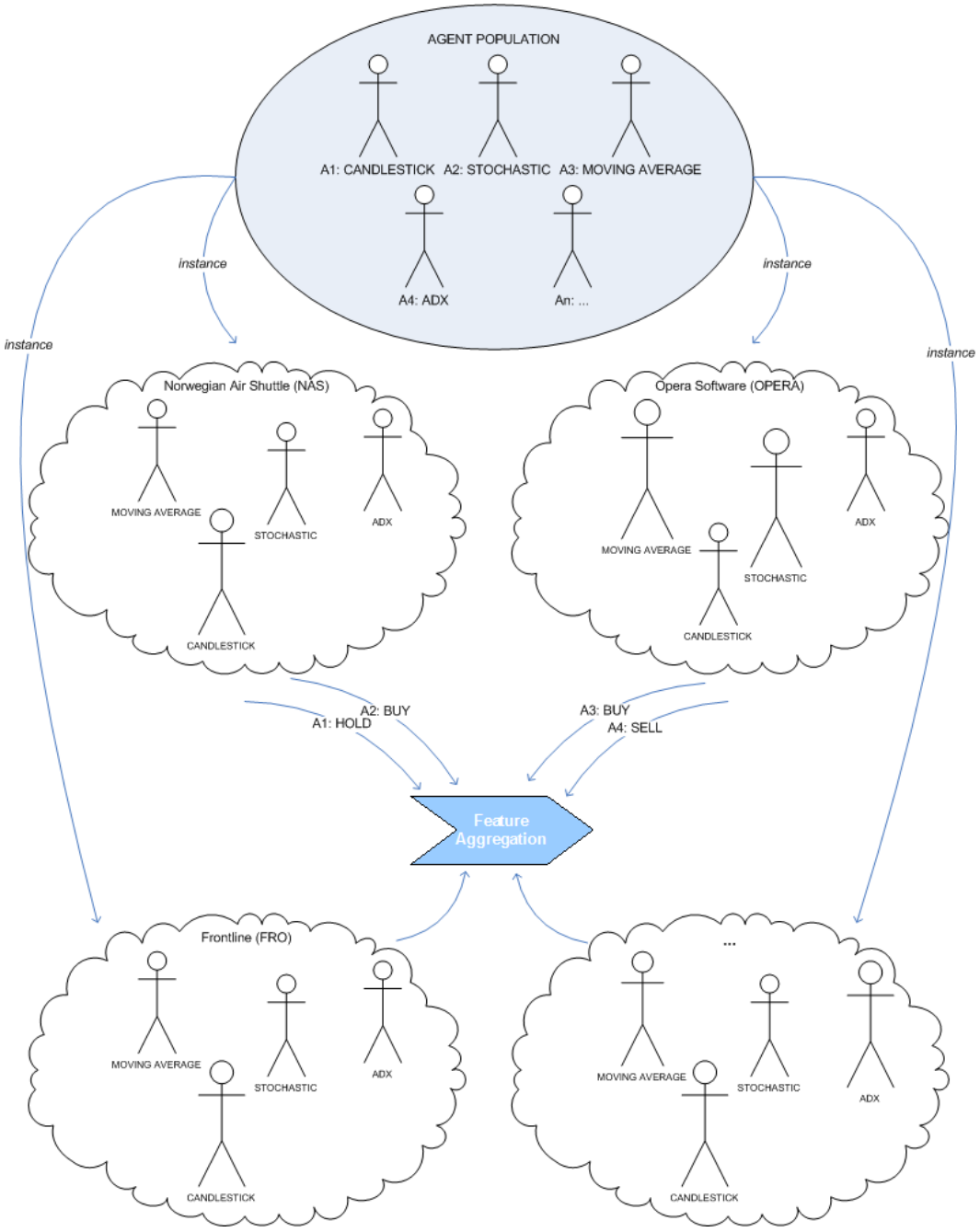


Figure 3.1: The Agent-Oriented Approach to Feature Generation

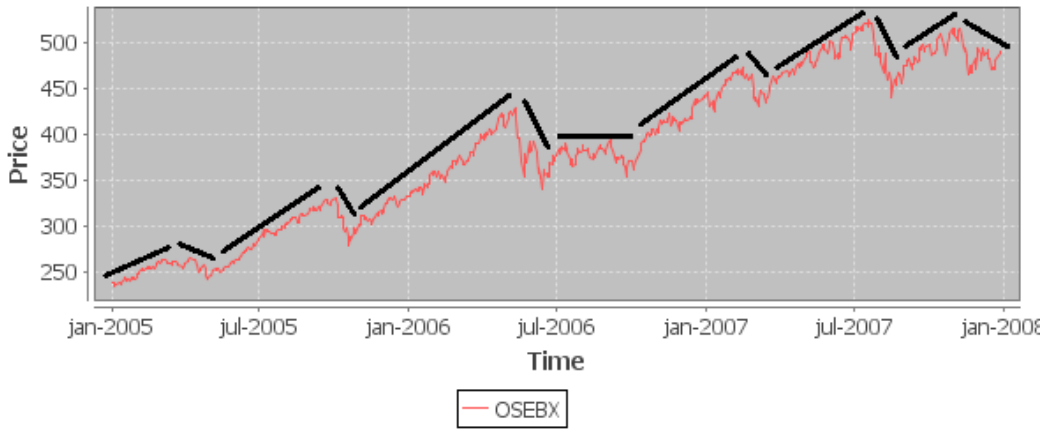


Figure 3.2: Oslo Benchmark Index (OSEBX) with manually drawn trend lines.

like to sell before the downtrend occurs and repurchase when the following uptrend starts to form. However, the trend identifying mechanism should not be overly sensitive to short-term fluctuations as that would result in falsely reporting a break in trend. Hence, the trend identifying mechanism should be able to detect several trends in a time series while not being overly sensitive to short-term fluctuations. In the following sections, three possible methods for calculating $\mathbf{G}$ and identifying trends are discussed and evaluated according to the above criteria.

3.1.1 Simple Linear Regression

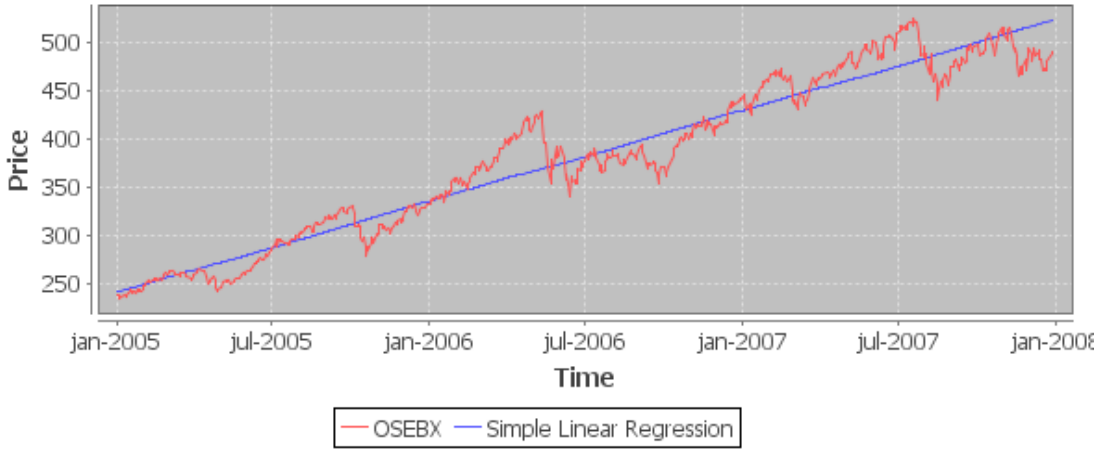


Figure 3.3: Simple Linear Regression

Simple linear regression is a statistical method for approximating a straight line through a set of n data points (Walpole et al., 2006). One assumes the presence of a straight line that describes the data, $y = \alpha + \beta x$, where the intercept α and slope β are calculated using the least squares estimator,

$$\sum_{X}([\hat{\alpha}x_i + \hat{\beta} - y_i]^2) \quad x_i \in X, y_i \in Y$$

where $\hat{\alpha}$ and $\hat{\beta}$ are estimators for α and β, respectively. The result is a trend line that describes the major trend in the data. A statistical hypothesis test can then be executed to determine if there is a significant linear relationship in the data,

$$\begin{aligned} H_0 : &\quad \beta = 0 \\ H_1 : &\quad \beta \neq 0 \end{aligned}$$

$$\text{Trend} = \begin{cases} \text{Uptrend} & \text{if } H_0 \text{ is rejected and } b > 0, \\ \text{Downtrend} & \text{if } H_0 \text{ is rejected and } b < 0, \\ \text{Notrend} & \text{otherwise.} \end{cases}$$

thereby determining if the data is, in fact, trending. A regression line for the Oslo Benchmark Index (OSEBX) is plotted in Figure 3.3. It is apparent that the regression line successfully identifies the major trend in the price data. However, given that the regression line is calculated as a straight line, it is not possible to identify the short-term downtrends from the regression line. This could be solved by using a higher-degree polynomial for the regression line. However, curvilinear regression can be computationally expensive to calculate and research show that curvilinear regression often leads to misleading results when applied to economic data (Leser, 1961). We thus conclude that we may use simple linear regression to determine major trends, but we need a different, more robust method, that will allow us to detect several trends in the price data.

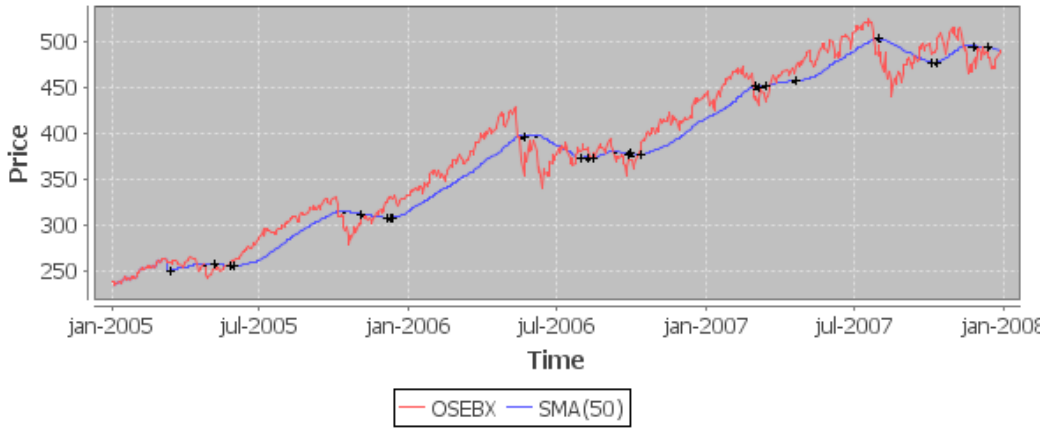
3.1.2 Moving Averages

Moving averages are running averages of a finite size window over a dataset that can be used as a trend-following device. A Simple Moving Average (SMA) of length n for a data point $x_i \in X$ is the unweighted mean of the n past data points,

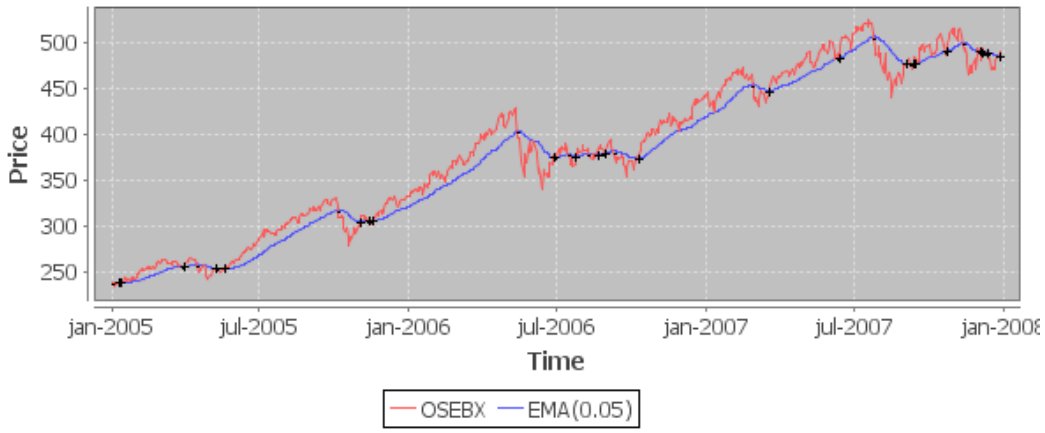
$$\text{SMA}_{x_i,n} = \frac{x_i + x_{i-1} + x_{i-2} + \cdots + x_{i-n-1}}{n} \quad x_i \in X$$

When a new observation becomes available, the oldest observation in the window is dropped and the new observation is added (i.e., the window slides over the new observation). When calculated over the entire data set $\mathbf{X}$ the SMA provides a smoothed representation of $\mathbf{X}$ where the level of smoothing is proportional to n, the length of the moving average.

A plot of an SMA with $n = 50$ days is given in Figure 3.4a where changes in the first difference of the SMA is marked with a "+" for changes from downtrend to uptrend and "-" for changes from uptrend to downtrend. The plot shows that the SMA successfully identifies the trend in intervals where the price moves in an orderly range either up or down (i.e., no large fluctuations). However, as each data point is equally weighted, single data points can have a disproportionately large effect on the trendline, causing the first



(a) $n = 50$



(b) $\alpha = 0.05$

Figure 3.4: Plot of the SMA and EMA. Changes in the first difference are denoted with "+" when the change is positive and "−" when the change is negative (i.e., changes from downtrend to uptrend and uptrend to downtrend, respectively).

difference to fluctuate in intervals where the price is somewhat volatile. Moreover, given that the SMA is calculated based on the past n data points, it is apparent that the trendline generated by the SMA is delayed by a factor proportional to n. This constitutes a problem as changes in trend will only be detected with significant delay. A shorter moving average would result in a shorter delay, but then at the cost of less smoothing and more fluctuations in the first difference. Both of these problems can be mitigated, but not eliminated, by a weighted moving average. The Exponential Moving Average (EMA) is a weighted moving average where each data point x_i is scaled by an exponential factor α,

$$\text{EMA}_{x_i,n} = \alpha \times x_{i-1} + (1 - \alpha) \times EMA_{x_{i-1}}$$

where EMA_{x_1} is typically set to x_1. Plot 3.4b shows that the EMA provides a smoother trendline that responds faster to trend shifts than the SMA. However, the trendline still suffers from some delay and is still fairly sensitive to sudden spikes and short-term fluctuations in the data. We thus conclude that moving averages can be used to identify several trends in a dataset, but the trendline generated by moving averages is too sensitive to short-term fluctuations.

3.1.3 Hodrick-Prescott Filter

The Hodrick-Prescott Filter is a mathematical tool used in macroeconomics to create a smoothed non-linear representation of a time series that is less sensitive to short-term fluctuations than long-term fluctuations. The Hodrick-Prescott filter assumes that any given time series $\mathbf{X}$ can be divided into a trend component τ_t and a cyclical component c_t and expressed by the sum $x_t = \tau_t + c_t$ (for our purpose c_t could perhaps better be described as a noise component, but we choose to remain consistent with standard notation). The cyclical component can then be obtained by subtracting τ from x giving $c_t = x_t - \tau_t$. The cyclical component c_t and the trend component τ_t can then be isolated by solving the following minimization problem,

$$\min_{\tau} \sum_{t=1}^{T} (x_t - \tau_t)^2 + \lambda \sum_{t=2}^{T} [(\tau_{t+1} - \tau_t) - (\tau_t - \tau_{t-1})]^2 \quad x_t \in X \tag{3.1}$$

where the first term of the equation is the sum of squared deviation of c_t and the second term is the first difference of the trend component. When solving the minimization problem the first term penalizes large values of c_t (i.e., poor fit) while the second term penalizes the lack of smoothness in τ_t. The trade off between the two terms is controlled by the λ parameter. Consequently, higher values of λ penalizes variations in the first difference of the trend component causing a smoother trend line that is less sensitive to short-term fluctuations than long-term fluctuations (it essentially controls the degree of smoothing over short-term fluctuations). Note that as λ approaches 0 the trend component approaches the original time series, and as λ approaches infinity τ_t approaches a linear trend. The data points in $\mathbf{X}$ are typically scaled with the natural logarithm before calculating τ.

Danthine and Girardin (1989) show that the solution to the minimization problem in Equation 3.1 is given by the matrix $\hat{\tau} = [\mathbf{I} + \lambda \mathbf{K}'\mathbf{K}]^{-1}\mathbf{x}$ where $\mathbf{x} = [x_1, \ldots, x_T]'$, $\tau = [\tau_1, \ldots, \tau_T]'$, $\mathbf{I}$ is a $T \times T$ identity matrix, and $\mathbf{K} = \{k_{ij}\}$ is a $(T-2) \times T$ matrix with elements given by

$$k_{ij} = \begin{cases} 1 & \text{if } i = j \text{ or } i = j+2, \\ -2 & \text{if } i = j+1, \\ 0 & \text{otherwise.} \end{cases}$$

By utilizing the fact that the matrix $[\mathbf{I} + \lambda \mathbf{K}'\mathbf{K}]^{-1}\mathbf{x}$ has a pentadiagonal structure fast numerical software packages can be used to efficiently solve Equation 3.1. This is important as the filter needs to be re-calculated on every new observation. We have adapted an algorithm developed by Kurt Annen[1] that efficiently calculates the Hodrick-Prescott Filter. The most apparent drawback of the filter is the problem of choosing an appropriate value for the the λ parameter. Figure 3.5 shows four plots of the HP filter with increasing λ values over the closing price of the Oslo Benchmark Index.

It is apparent from Figure 3.5 that the filter works according to its purpose; it is less sensitive to short-term fluctuations than long-term fluctuations. We also see that increasing values of the λ parameter causes greater smoothing in the generated trendline. If we examine the plot with $\lambda = 5000$ we find that it perfectly correspond to the manually drawn trendlines given in 3.2. We thus choose to use the HP Filter with $\lambda = 5000$ as the trend identifying mechanism in the Trend Agent, which is then defined as follows,

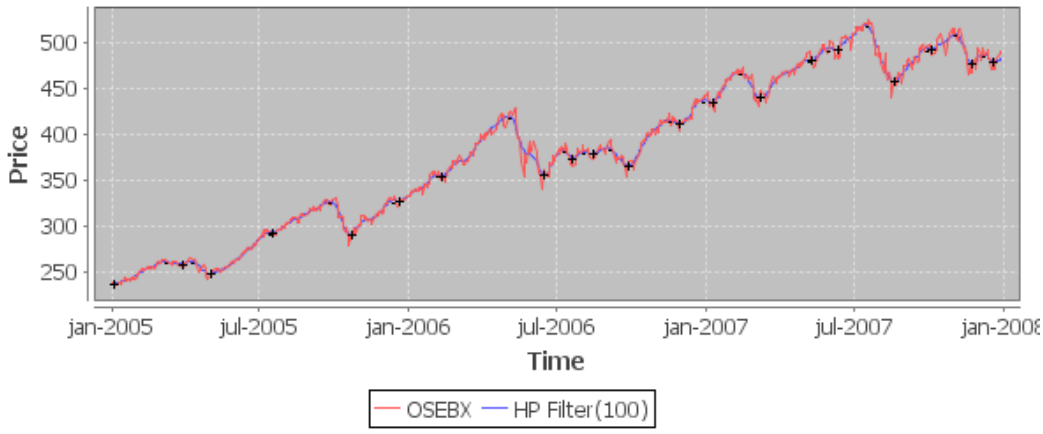
$$\text{Trend-Agent}_n(P_s, t) \rightarrow \{\text{Uptrend, Downtrend, Notrend}\}$$

The Trend Agent takes as input P_s, the price history for some stock, a trend length size n, a time index t, and returns the trending direction of the closing prices of P_s in the interval from $P_s(t-n)$ to $P_s(t)$. This is done by first calculating the Hodrick-Prescott Filter for $P_s(1)$ to $P_s(t)$. If the sign of the first difference of the trendline generated by the HP Filter is constant and positive in $P_s(t-n)$ to $P_s(t)$ Uptrend is returned, if constant and negative Downtrend is returned, or Notrend if it is not constant.

3.2 Moving Average Crossover Agent

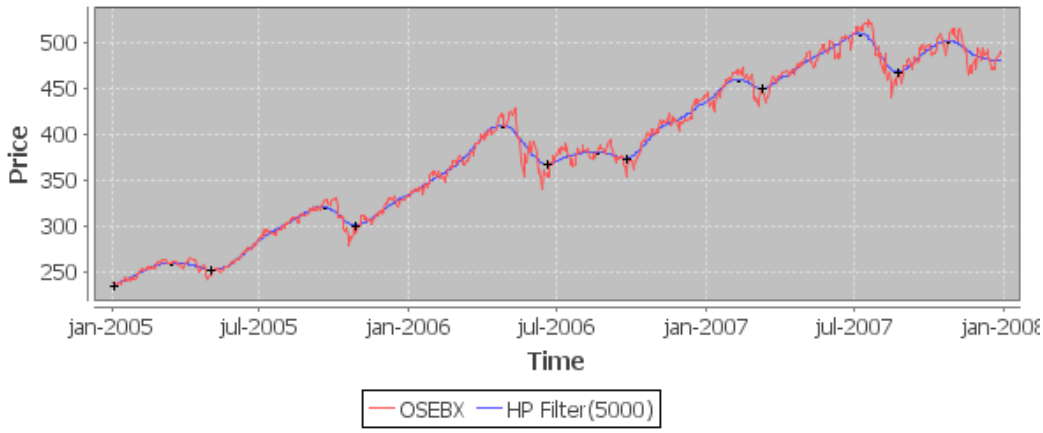
The Moving Average Crossover Agent (MAC) is a simple agent that detects crossovers of two moving averages. Moving average crossovers is a popular technical trading rule used to detect trend reversals (thereby generating buy and sell signals) (Turner, 2007). Two simple moving average lines with different lengths, n and m where $n < m$, are plotted simultaneously and buy and sell signals are generated at points where the two moving averages intersect. Buy signals are generated when the shorter moving average (i.e., of length n) rises above the longer moving average (i.e., of length m), and sell signals are generated when the shorter moving average falls below the longer moving average.

[1] `http://www.web-reg.de`



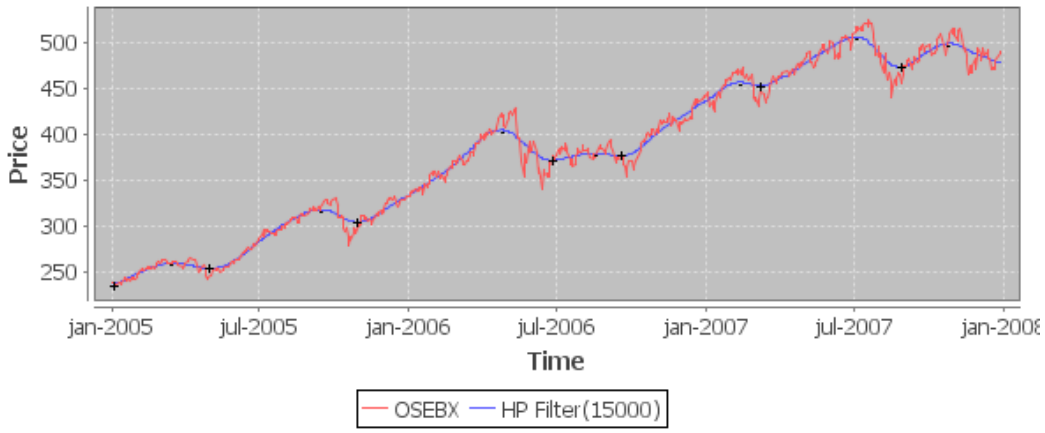
(a) $\lambda = 100$

(b) $\lambda = 1000$



(c) $\lambda = 5000$

Figure 3.5: Continued on next page...



(d) $\lambda = 15000$

Figure 3.5: Hodrick-Prescott Filter with increasing values for the λ parameter. Changes in the first difference are denoted by "+"/"-".

The general idea is that each moving average measures the trend at different time scales. Hence, when a shorter moving average rises above a longer moving average, the theory states that the long-term trend as measured by the long moving average may reverse to the short-term trend as measured by the short moving average. Consequently, a buy signal is generated and hold signals are generated until the short moving average eventually drops below the long moving average. The assumption is that buying pressure is strong at the crossover point because the average closing price over the past n days has exceeded the average closing price over the past m days. For example, if $n = 20$ and $m = 50$ and the n-line rises above the m-line the average closing price over the past 20 days is now higher than the average closing price over the past 50 days, signifying that buying pressure has increased and that there may be a trend reversal to the trend measured by the n-line. If a trend reversal does occur, the m-line will eventually follow in the direction of the n-line. If the trend subsequently reverses again (after some time), the n-line will drop below the m-line and a sell signal will be generated.

The Moving Average Crossover Agent (MACA) is initialized with a length for the short and long moving average, respectively. It then calculates two simple moving averages, one for each length. Buy and sell signals are then generated by the following rules,

$$MAC_{n,m}(t) = \begin{cases} Buy & \text{if } SMA_n(t-1) < SMA_m(t-1) \text{ and } SMA_n(t) > SMA_m(t), \\ Sell & \text{if } SMA_n(t-1) > SMA_m(t-1) \text{ and } SMA_n(t) < SMA_m(t), \\ Hold & \text{otherwise.} \end{cases}$$

where $n < m$. Figure 3.6 shows a plot of the signals generated by two separate instances of the agent with different moving average length pairs, one short pair ($n = 5$, $m = 20$) and one long pair ($n = 20$, $m = 50$). As seen by the plot, the crossovers do generate some seemingly profitable signals. However, as each signal is delayed, sharp price movements cause many signals to be generated at less profitable positions. This is immediately apparent by the sell signal generated in late 2005. Here, the price dropped sharply and a

sell signal is generated at the end of the downtrend. For the shorter pair instance we see that the effect is less negative than for the longer pair instance. Moreover, in situations where the price is moving sideways, whiplashes occur in the moving averages and many spurious signals are generated. This can be viewed from July, 2006 to January, 2007. This effect is less dramatic for the longer pair instance than the shorter pair instance. Thus, there is a trade-off in using a short-pair instance versus a long-pair instance. Shorter lengths will generate signals with less delay, but more spurious signals will be generated when the price is moving sideways. Longer lengths will generate signals with more delay, but less spurious signals are generated when the price is moving sideways. This result is one of the motivating factors for the *Feature Aggregation* module presented in Chapter 4. By placing several instances of the Moving Average Crossover Agent in context with other agents, spurious signals may be mitigated and the benefit of each length pair can be harvested.

(a) $n = 5$, $m = 20$

(b) $n = 20$, $m = 50$

Figure 3.6: Signals generated by moving average crossovers. Buy signals are indicated with a "+" and sell signals are indicated with a "−".

Several studies have been made to assess the predictability of trading strategies that use moving averages (Brock et al., 1992; Gencay, 1996, 1998; Gencay and Stengos, 1998).

Brock et al. (1992) use moving average crossovers of several different lengths to produce buy and sell signals for the Dow Jones Index from 1897 to 1986. Their results provide strong statistical support in favor of moving average crossovers (Larsen, 2009).

3.3 Candlestick Agent

Candlestick pattern analysis is a technique used in technical analysis to predict short-term trend reversals (Murphy, 1999). Candlestick patterns are formations of typically one to three candlesticks. Remember from Section 2.1.4 that a candlestick is simply a charting tool that includes all price information available for a stock in a given time frame (Figure 3.7). The Candlestick Agent holds a library of candlestick patterns that, when detected in the price history for some stock, generate buy and sell signals depending on the detected pattern.

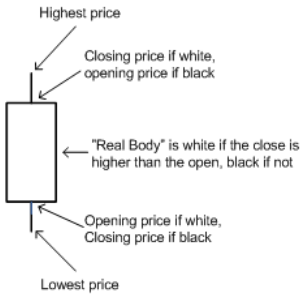
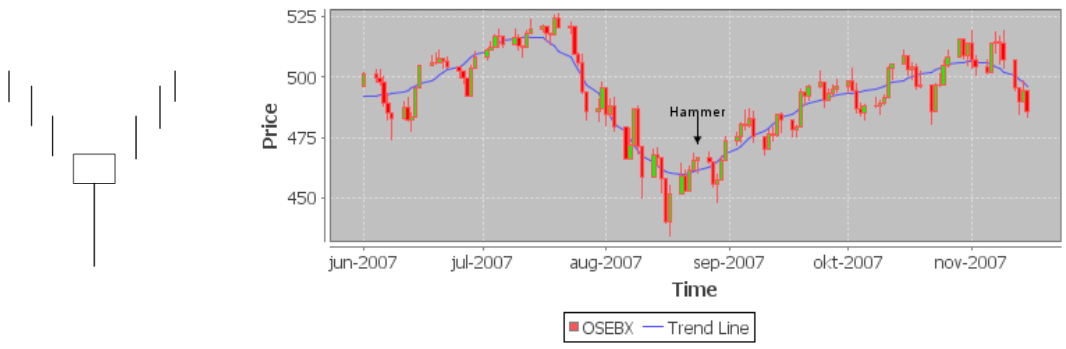


Figure 3.7: Candlestick

3.3.1 Candlestick Patterns

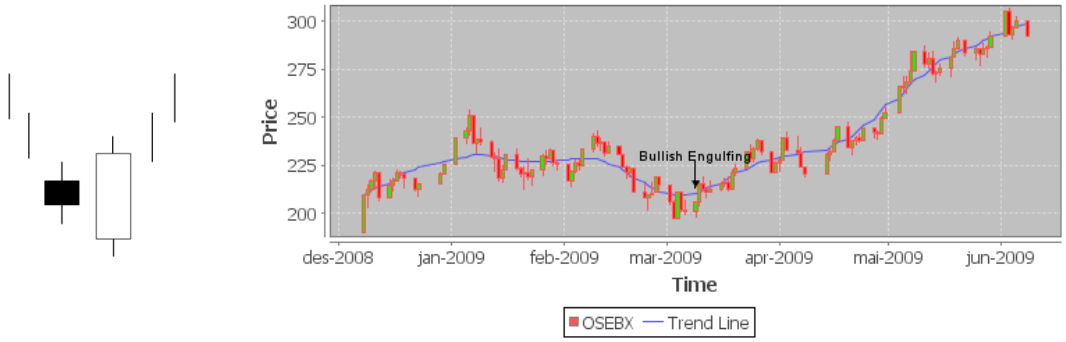
As mentioned, candlestick patterns are formations of one to several candlesticks that are designed to predict short-term trend reversals. As such, each candlestick pattern needs to occur in a discernible trend to be predictive. The Candlestick Agent holds a library of 12 candlestick patterns where 6 of which predict and upcoming uptrend (i.e., they need to occur in a downtrend) and 6 predict an upcoming downtrend (i.e., they need to occur in an uptrend). Patterns that predict an uptrend generate buy signals, and patterns that predict a downtrend generate sell signals. In the following, two patterns contained in the library of the Candlestick Agent are described.



(a) (b)

Figure 3.8: The Hammer Pattern. Figure 3.8a shows a schematic view of the pattern while Figure 3.8b shows an actual instance of the pattern in the price history for the Oslo Benchmark Index (OSEBX).

Figure 3.8a shows a schematic diagram of the *Hammer pattern*, a simple candlestick pattern composed of a single candlestick. The candlestick has a white real body at the top of the candlestick and the length of the lower shadow is twice the length of the real body. When the pattern forms in a downtrend the long lower shadow (i.e., the range between the opening price and lowest price) indicates that the stock experienced strong selling pressure within the day, pulling the price down and thereby confirming the downtrend. However, the white real body at the top of the trading range shows that the stock closed near its high, indicating that the day was concluded with strong buying pressure. Pressure has thus shifted from sellers to buyers (interpreted as a shift in supply and demand), indicating that the preceding downtrend may reverse to an uptrend. The theory states that this shift in control from sellers to buyers that give the pattern its suggested predictive effect. Figure 3.8b shows an actual occurrence of this pattern as detected by the Candlestick Agent. In this particular case, the pattern produced a positive returns investment opportunity.



(a) (b)

Figure 3.9: The Bullish Engulfing Pattern. Figure 3.9a shows a schematic view of the pattern while Figure 3.9b shows an actual instance of the pattern in the price history for the Oslo Benchmark Index (OSEBX).

In Figure 3.9 a candlestick pattern named the *Bullish Engulfing pattern* is given. The pattern is composed of two candlestick where the latest candlestick engulfs (i.e., covers) the real body of the previous candlestick. The first black candlestick confirms the current downtrend, but as the second wide-range white candlestick is formed and covers the first candlestick, buying pressure has exceeded selling pressure (i.e., the stock opens lower than the previous day's close but closes higher than the previous day's open) and the current downtrend may reverse to an uptrend. In Figure 3.9b we see the pattern appear in the price history for OSEBX.

As the discussion above shows, the rationale for candlestick patterns is based on their interpretation as shifts in buying and selling pressure. Some practitioners view the candlestick patterns as detectors of market psychology that gauge the mind of the traders causing the price action (Murphy, 1999). Although practitioners of candlestick analysis vividly portray their utility (Murphy, 1999; Nison, 1991; Turner, 2007), there seems to be limited statistical research available on the actual predictability of candlestick patterns (Park and Irwin, 2007). However, Caginalp and Laurent (1998) did a study on a selection of 10 different candlestick patterns applied to the S&P stocks (a Benchmark

Index of 500 large American companies) between 1992 and 1996. Their results indicate that candlestick pattern can create substantial profits compared to a simple buy-and-hold strategy (where buy signals are created randomly and stocks are held for a fixed number of days). Specifically, bullish reversal patterns, like the Hammer and Bullish Engulfing patterns explained above, produce an average return of 0.9% during a 2-day holding period for the S&P 500 stocks over 1992-1996. This is a signifcant result as each investment dollar is committed for an average of only two days, which would result in an annual return of 309% of the initial investment (Caginalp and Laurent, 1998).

3.3.2 Modeling the Candlestick Patterns

The candlestick library employed by the Candlestick Agent is modeled using a rule-based approach. Each candlestick pattern is defined by a set of constraints on one or more candlestick attributes. The candlestick attributes include the open, close, high and low prices, as well as the following auxiliary attributes used to easy the definition of some patterns. Each attribute is derived from the price and are given as a percentage of the price range.

$$\text{Color} = \begin{cases} \text{White} & \text{if Close} > \text{Open} \\ \text{Black} & \text{if Close} < \text{Open} \end{cases}$$

$$\text{Range} = \text{High} - \text{Low}$$

$$\text{Body} = \frac{|\text{Close} - \text{Open}|}{\text{Range}}$$

$$\text{Upper-Shadow} = \begin{cases} (\text{High} - \text{Close})/\text{Range} & \text{if Color} = \text{White} \\ (\text{High} - \text{Open})/\text{Range} & \text{if Color} = \text{Black} \end{cases}$$

$$\text{Lower-Shadow} = \begin{cases} (\text{Open} - \text{Low})/\text{Range} & \text{if Color} = \text{White} \\ (\text{Close} - \text{Low})/\text{Range} & \text{if Color} = \text{Black} \end{cases}$$

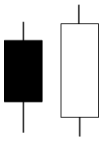
Furthermore, each candlestick pattern is associated with an attribute that specifies a previous trend constraint. For example, Hammer pattern is only predictive if it occurs in a downtrend. When the Candlestick Agent finds an instance of the Hammer pattern it consequently polls the Trend Agent to determine if the instance occurred in a downtrend. Instances where the actual trend is inconsistent with the trend constraint are disregarded.

The constraints defining the Hammer pattern is given in Figure 3.10a. Here, C_t refers to a candlestick at time t, and $C_{t,a}$ refers to the value of attribute a for candlestick C_t. In cases where the pattern is composed of several candlesticks, C_t refers to the latest candlestick in the pattern, and C_{t-n} refers to the candlestick n time steps prior to C_t. In accordance with the description of the Hammer pattern, the second rule in Figure 3.10a specifies that the candlestick should be white, the third rule specifies that the lower shadow should be twice the length of the body, the fourth rule specifies that the upper

shadow should be short (i.e., less than 5% of the total range), and the last rule specifies that the candlestick should have some body. In Figure 3.10b, the rules defining the bullish engulfing pattern are given. Here, rule 4 and 5 specify the engulfing constraint (i.e., the second body should cover the first body). Each pattern implemented by the agent is given in Appendix B.

1. Trend = Downtrend
2. $C_{t,\text{Color}} = \text{White}$
3. $C_{t,\text{Lower-Shadow}} \geq 2 \times C_{t,\text{Body}}$
4. $C_{t,\text{Upper-Shadow}} \leq 0.05$
5. $C_{t,\text{Body}} \geq 0.10$

(a) Hammer Pattern



1. Trend = Downtrend
2. $C_{t-1,\text{Color}} = \text{Black}$
3. $C_{t,\text{Color}} = \text{White}$
4. $C_{t,\text{Open}} \leq C_{t-1,\text{Close}}$
5. $C_{t,\text{Close}} \geq C_{t-1,\text{Open}}$

(b) Bullish Engulfing Pattern

Figure 3.10: Rule-sets for the Hammer and Bullish Engulfing Pattern

We can then define the Candlestick Agent as follows,

$$\text{Candlestick-Agent}(P_s, t) \rightarrow \{\text{Buy, Sell, Hold}\}$$

The Candlestick-Agent takes the price history P_s for a particular stock, a time index t and iterates over the rule-sets contained in the candlestick pattern library and returns Buy, Sell or Hold depending on whether or not it found a rule set that matched at time t. A Buy signal is generated if the agent detects a pattern that predicts an upcoming uptrend, and a sell signal if it detects a pattern that predicts an upcoming downtrend. Hold is returned if none of the rule-sets match.

3.4 Stochastic Agent

The Stochastic Agent generates buy and sell signals by a set of heuristics on values generated by a momentum indicator called the Stochastic Oscillator. Momentum indicators are a class of tools that are used measure trend momentum and to detect situations when a stock is overbought or oversold (Murphy, 1999). An overbought stock is a stock whose price has increased drastically over a short period of time and is trading at an artificially high price compared to recent price activity. If a stock is trading at an artificially high price the theory suggests that the price may reverse, thereby generating a sell signal. Similarly, an oversold stock is a stock whose price has fallen drastically over a short period of time, a situation that may indicate that the price will reverse to the upside, thereby generating a buy signal. When a stock is overbought it is usually a result of unjustifiably high demand, which means that the stock is probably overvalued and economic theory tells us that the market will soon adjust and the stock will probably experience a pullback. An oversold stock, however, usually indicates that the stock is undervalued (fear, or panic selling typically result in an oversold stock) which may signal a good short-term investment opportunity (Turner, 2007).

3.4.1 Stochastic Oscillator

The Stochastic Oscillator is based on the observation that closing prices tend to close to the upper end of the recent price range in uptrends and to the lower end in downtrends (Murphy, 1999). It is calculated using two lines, the main $\%K$ line and the $\%D$ signal line. The $\%K$ line is calculated over some n-length period by the following formula,

$$\%K_n(t) = 100\frac{C_t - L_n}{H_n - L_n}$$

where C_t is the latest closing price in the period, L_n is the lowest low in the period, and H_n is the highest high in the period. The $\%K$ line is typically calculated over the past 14 days, but other time frames may also be used. If we examine the above formula we see that the $\%K$ line is basically a percentage measure of where the current closing price is in relation to the total price range over the past n days. For example, a 14-day $\%K$ value of 20 would indicate that the latest closing price in the price range is 20% above the lowest low and 80% below the highest high the past 14 days. High values of $\%K$ is typically interpreted as an overbought signal as the closing price is then near the top of price range. Similarly, low values are intepreted as an oversold signal as the closing price is near the bottom of the total price range.

The signal line is either a 3-day simple moving average of the $\%K$ line called the $\%D_{\text{fast}}$ line, or another 3-day simple moving average of the $\%D_{\text{fast}}$ line creating a smoother line called the $\%D_{\text{slow}}$,

$$\%D_{\text{fast}} = SMA_3(\%K)$$
$$\%D_{\text{slow}} = SMA_3(\%D_{fast})$$

There are several ways of interpreting the values produced by the Stochastic Oscillator. The approach employed by the Stochastic Agent is adapted from (Murphy, 1999) where values of $\%K$ above 80 are interpreted as the overbought range and values below 20 are interpreted as the oversold range. A buy signal is then generated if the $\%K$ line crosses above the $\%D_{\text{slow}}$ line in the oversold range (that is, the crossover occurs when both lines are below 20). Similarly, a sell signal is generated if the $\%K$ line crosses below the $\%D_{\text{slow}}$ line in the overbought range (that is, the crossover occurs when both lines are above 80). The Stochastic Agent is thus defined as follows,

$$\text{Stochastic}(P_s, t) = \begin{cases} \text{Buy} & \text{if } \%K(t-1) < \%D(t-1) < 20 \text{ and } \%D(t) < \%K(t) < 20, \\ \text{Sell} & \text{if } \%K(t-1) > \%D(t-1) > 80 \text{ and } \%D(t) > \%K(t) > 80, \\ \text{Hold} & \text{otherwise.} \end{cases}$$

Figure 3.11 shows a plot of the signals produced by the Stochastic Agent along with a plot of the signal line and main line. The plot shows that the agent correctly identifies the stochastic crossovers. Analyzing the utility of the stochastic signals will be done in Chapter 6.

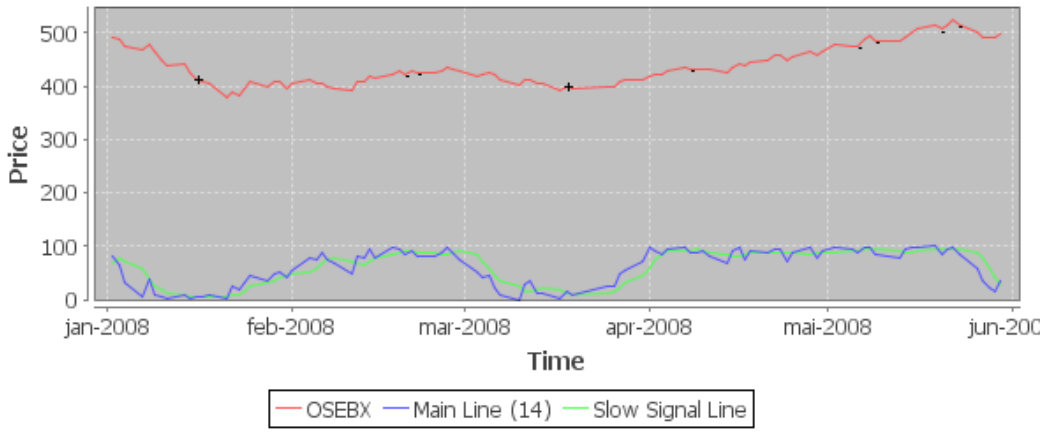


Figure 3.11: Signals generated by the Stochastic Agent over a short period of OSEBX. Buy and sell signals are marked with '+'/'-', respectively.

3.5 Volume Agent

Volume refers to the total number of shares traded of a stock in any given time frame, an important source of information for several reasons. For one, it is the only source of information that we use which is not a derivative of price, and as such it provides a "second opinion" on price movements (Turner, 2007). Secondly, volume information allows us to measure the strength and value of price movements, an important factor when analyzing other technical indicators. For example, the oversold and overbought signals discussed in the previous section are often accompanied and confirmed by high volume, robust uptrends are typically accompanied by steady volume (indicating that if volume becomes unstable in an uptrend we might have a trend reversal), candlestick patterns are typically only predictive if accompanied by high volume, etc. The dataset available to the Volume Agent includes the total number of shares traded each day for each stock in the dataset.

Turner (2007) formulates the volume rule that is stated as follows, *"when volume expands, price expands higher or lower; when volume contracts, price contracts"*. In other words, price often follow volume, and volume often follow the price trend (Linløkken and Frölich, 2004). When this is not the case, volume expands and price does not, it is referred to as a price-volume divergence, which often signify that volume will expand in upcoming price movements. When a trend reversal occurs, for example from a downtrend to an uptrend, volume gives a perspective on the strength of the reversal. If the reversal is accompanied by high volume (compared to previous volume) the trend reversal is likely to hold and manifests the new trend. A primary objective thus becomes to measure the level of recent volume compared to previous volume. That is, how strong is the volume (interest) in the stock today compared to previous days. A simple method used to measure relative volume strength is by a volume oscillator (Turner, 2007).

The Volume Agent employs a volume oscillator that use two simple moving averages with different lengths over the volume data. If volume as measured by the short moving average

is higher than volume as measured by the long moving average, the short-term volume trend is higher than the long-term volume trend, and we generate a Strong-Volume signal. As such, it signals where the short-term current volume is in relation to average volume in the past. The Volume Agent is thus defined as follows,

$$\text{Volume-Agent}_{n,m}(V_s, t) = \begin{cases} \text{Strong-Volume} & \text{if } \text{SMA}_n(t) > \text{SMA}_m(t), \\ \text{Weak-Volume} & \text{otherwise.} \end{cases}$$

where $n < m$ and V_s is the volume history associated to stock s. For example, if $n = 2$ and $m = 14$ days (the default values), a Strong-Volume signal for some day t would indicate that the volume averaged over the past two days is higher than the volume averaged over the past 14 days, which is an important signal in combination with other signals.

3.6 ADX Agent

The ADX Agent implements a measure of *trend strength* through the use of the Average Directional Index (ADX). The ADX measure the degree to which the price is trending, but provides no information on the direction of the trend. It should thus be very useful in relation to the Trend Agent. The calculations used by the ADX are fairly involved compared to the other indicators we have discussed so far. The ADX is calculated by smoothing another indicator, the directional movement index (DX), with an exponential moving average over some time period n (typically 14 days). The DX is, in turn, calculated by combining and smoothing two other indicators, the positive and negative directional indicator (DI^+ and DI^-), over the same time period. DI^+ and DI^- are, in turn, calculated based on a notion of positive and negative directional movement (DM^+ and DM^-) which is defined as the largest part of the current trading range that is outside the previous day's trading range (Wilder, 1978),

$$\text{DM}_t^+ = \begin{cases} \max(\text{high}_t - \text{high}_{t-1}, 0) & \text{if } \text{high}_t - \text{high}_{t-1} \geq \text{low}_t - \text{low}_{t-1}, \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{DM}_t^- = \begin{cases} \max(\text{low}_t - \text{low}_{t-1}, 0) & \text{if } \text{high}_t - \text{high}_{t-1} \leq \text{low}_t - \text{low}_{t-1}, \\ 0 & \text{otherwise.} \end{cases}$$

In Wilder (1978) directional movement is defined absolutely as being either up or down, it cannot be a combination of both. That is, if the largest part of the current trading range is above the previous day's trading range, directional movement is positive and DM^+ is equal to the current high minus the previous high. If the largest part of the current trading range is below the previous day's trading range, directional movement is negative and DM^- is equal to the current low minus the previous low. In other words, DM^+ measures the size of any up move for today compared to yesterday and if $DM_t^+ > 0$ then the price has moved up on day t. Similarly, DM^- measures the size of any down move for today compared to yesterday and if $DM_t^- > 0$ then the price has moved down on day t. Directional movement is then defined as a function of range which results in the

positive and negative directional indicator for a point t which are defined as follows,

$$\mathrm{DI}_t^+ = \mathrm{DM}_t^+ / \mathrm{TR}_t$$

$$\mathrm{DI}_t^- = \mathrm{DM}_t^- / \mathrm{TR}_t$$

where TR_t is the true range for day t. The trading range for a single day is typically defined as $\mathrm{high}_t - \mathrm{low}_t$. The true range extends this concept to yesterday's closing price if it was outside today's range, $\mathrm{TR}_t = \max(\mathrm{high}_t, \mathrm{close}_{t-1}) - \min(\mathrm{low}_t, \mathrm{close}_{t-1})$. DI_t^+ then essentially defines the percent of the true range that is up for time t (if any) and DI_t^- is the percent of the true range that is down for time t (if any). DI^+ and DI^- are then calculated and smoothed over the past n days to produce two directional indicator lines,

$$\mathrm{DI}_n^+ = \mathrm{EMA}(\mathrm{DI}^+, n)$$

$$\mathrm{DI}_n^- = \mathrm{EMA}(\mathrm{DI}^-, n)$$

When calculated over $n = 14$ days $\mathrm{DI}_{14}^+ = 0.36$ would indicate that 36% of the true range for the past 14 days was up. The same holds for down direction with DI^-. We can then calculate the directional movement index (DX) for time t by combining the directional indicators,

$$\mathrm{DX}_t = ((\mathrm{DI}_t^+ - \mathrm{DI}_t^-)/(\mathrm{DI}_t^+ + \mathrm{DI}_t^-))$$

which represents the true directional movement. That is, each day with positive directional movement we are adding to DI^+ and subtracting from DI^-, resulting in a high value of DX_t. If the price moves in a sideways direction, the difference between DI^+ and DI^- will be small which is interpreted as non-directionality.

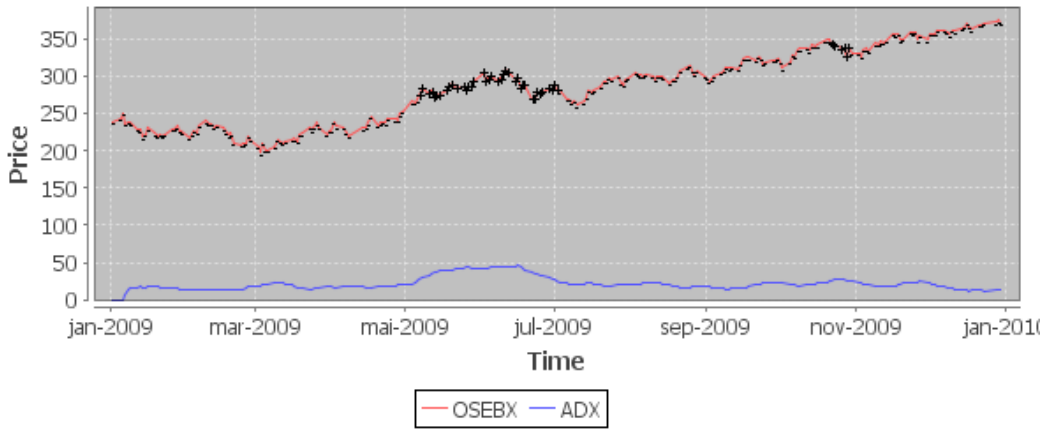


Figure 3.12: ADX Plot for OSEBX over 2009.

Thus, high values of DX indicate much directional movement while low values of DX indicate little directional movement. In order to make the DX more reliable it is smoothed

using an exponential moving (reducing the effect of high price fluctuations) resulting in the Average Directional Index,

$$\text{ADX}_n = 100 \times \text{EMA}(\text{DX}, n)$$

In general, low values (i.e., less than 25) of the ADX are interpreted as the price being in a weak trend and high values (i.e., above 25) are interpreted as the price being in a strong trend. The rationale behind this choice is difficult to explained due to the involved calculations in the ADX, and we rather refer the interested reader to (Wilder, 1978) for a more in-depth analysis of the directional movement system. Figure 3.12 shows a plot of the ADX-line over the Oslo Benchmark Index (OSEBX). ADX values below 25 are marked with a '-' to indicate weak trend and '+' to indicate strong trend.

4 Feature Aggregation

The domain knowledge implemented in the feature generation module is not intended to be used as the sole basis for making investment decisions (Murphy, 1999; Turner, 2007). Technical traders evaluate different indicators in different situations and different contexts as each indicator provides a different perspective on the price data. Moreover, technical traders find that certain indicators provide desired results on only a subset of stocks, with some indicators generating successful results on one stock but failing on another. Measures of trend and volume are often viewed as important indicators in context of other trading rules, and are typically used to confirm or disregard trade signals produced by other indicators (Linløkken and Frölich, 2004). Technical traders thus have another form of knowledge that they use to evaluate when and in what circumstances the different indicators are appropriate. Due to the number of combinations and different contexts the indicators can be used, formalizing this knowledge is practically impossible, and as Turner (2007) points out, is a knowledge that can only be acquired through experience. Hence, there seems to be no sound analytical algorithm that can be used to determine how to evaluate the different features, thus motivating the idea of a second reasoning layer.

The *Feature Aggregation* module employs machine learning to aggregate and place the generated features in context, creating a unified (and hopefully) profitable investment strategy. The work presented in this chapter is consequently focused on the implementation of a *learning module* that use price data to learn how to utilize the domain knowledge implemented in the feature generation module. We begin by defining the learning problem.

4.1 The Learning Problem

When designing a system that is intended to improve its performance through experience, a typical approach is to begin by formalizing the problem as a well-defined learning problem. This is done by first identifying the task that the system will execute and learn, a performance measure that evaluates how well the system performs at the designated task, and the experience (i.e., training data) that will be used by the system to learn the task (Mitchell, 1997). In our case, the task involves learning how to best combine and contextualize the feature-values produced by the agents into a unified investment strategy. This can be formalized by a target function,

$$\text{Invest}(A(P_s), t) \rightarrow \{\text{Buy, Sell, Hold}\} \tag{4.1}$$

where A is a set of agent instances from the agent population and $A(P_s)$ are the trade signals generated by the agents in A for price history P_s. Given some time index t,

the target function should return an investment decision (e.g., buy the stock, sell the stock or hold the stock). As such, the target function represents the target concept that the system is intended to learn, an investment strategy for a stock based on the agents described in the previous chapter. Choosing a representation for the target function can be a non-trivial task. Possible alternatives include a decision tree, neural network, rule-engine, etc. It is non-trivial not just because the choice of representation should be appropriate according to the problem domain and required input and output, but the choice also dictates the type of learning algorithm employed. The choice of representation for the target function is discussed further in Section 4.2.

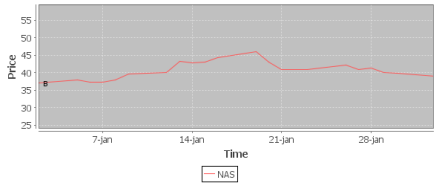
The training data available to learn Invest includes the daily price history for all stocks currently listed on the Oslo Benchmark Index. The price history includes all daily price levels (i.e., open, close, high and low) as well as the traded volume each day. In machine learning, there are two primary types of learning distinguished by the type of feedback available to the learning system, namely supervised and unsupervised. In the supervised case, the training data includes the correct output in every training example. That is, every output produced by the learned function can be instantly evaluated by comparing it to the correct output given in the training data. This is the traditional approach used in most industrial applications of machine learning. In the unsupervised case, no specific output is available in the training data. Training data consisting of daily stock prices is essentially unsupervised data as there is no information in the data that will allow us to locally (and instantly) evaluate every output of the learned function. However, we argue that for our purpose of learning an investment strategy, it is neither beneficial nor appropriate to locally evaluate every output of Invest.

For example, consider a plot of the price history for Norwegian Air Shuttle (NAS) given in Figure 4.1a. Here, two potential trade signals have been marked on the plot. Looking at Figure 4.1a we see that the signals are generated at profitable positions with the buy signal catching the strong uptrend at the start of the period and the sell signal catching the downtrend at the end of the period. However, by examining the signals locally at a closer interval, given in Figure 4.1b and 4.1c, the signals seem less profitable and would consequently not be credited according to their true utility if evaluated locally. As a result, it seems more appropriate to view the task of learning Invest as an optimization problem where the overall performance of the learned function is evaluated in retrospect rather than instantly for every output.

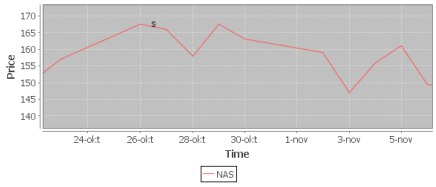
This complicates both the learning algorithm employed and the performance measure used. The performance measure will essentially have to use a global evaluation criteria that evaluates the combined utility of every output of Invest over the entire training data. Given that the target function is interpreted as an investment strategy and the preceding discussion, we define the performance measure as the profit-ratio generated by a portfolio simulation over trade signals generated by Invest for some price history P given an initial investment capital I. As the performance measure will be used to evaluate other parts of the system as well, we define it generally for some prediction model M that generates trade signals {Buy, Sell, Hold} over price history P_s,

$$\text{Performance}(M, I, P_s) = \frac{\text{Portfolio-Value}_1}{\text{Portfolio-Value}_T} \tag{4.2}$$

(a) Price history for NAS (Norwegian Air Shuttle)



(b) Buy Signal Local



(c) Sell Signal Local

Figure 4.1: The price history for NAS with a potential buy signal denoted by a ”B” and sell signal denoted by an ”S”.

where Portfolio-Value$_1$ and Portfolio-Value$_T$ is the total value of the simulated portfolio at the start and end of the simulation, respectively. The performance measure thus iterates over $t = [1, \cdots, T]$, retrieves a trade signal from the prediction model by evaluating $M(P_s(t))$ for each t and simulate transactions based on the generated trade signal and associated price history. The output is denoted the profit-ratio of the simulation, i.e., the initial investment capital divided by the total value of the portfolio at the end of the simulation. Our goal with the learning algorithm is then to find a function Invest that will maximize Performance. The design of the portfolio simulation is discussed in greater detail in Chapter 5. In the following sections the choice of representation for the target function and the algorithm implemented to learn the target function will be explained.

4.2 Agent Decision Trees

In order to successfully apply the domain knowledge implemented in the feature generation module we need to aggregate and place the generated features in context of each other. That is, we need to create a classification model that classifies different combinations of the generated features according to the target function given in Equation 4.1. We consequently need a data structure that aggregates the different features and assigns classifications to different feature-value combinations. Decision tree classification is a framework that facilitates the preceding requirements. Decision Tree Learning (DTL) is

a popular machine learning technique typically used for learning discrete-value target functions where the problem can be described by a finite set of feature-value pairs. It has been successfully applied to a whole range of practical problems such as learning to diagnose medical cases, learning to determine equipment malfunctions and learning to evaluate the credit risk of loan applicants (Mitchell, 1997).

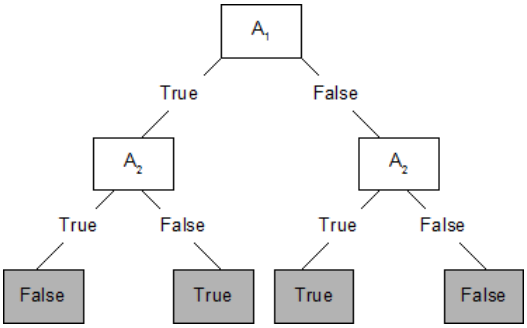


Figure 4.2: Decision Tree for $A_1 \oplus A_2$.

In a decision tree the problem attributes are organized in a tree structure where each internal node is assigned to an attribute with outgoing arcs corresponding to the possible values of the attribute. Possible output values for the target function (i.e., classifications) are added as leafs called classification nodes. Figure 4.2 shows a decision tree that represents a boolean function with two two boolean attributes A_1 and A_2. Classification nodes are marked by a gray box in the example. A classification is reached by traversing the tree, testing the attribute at each associated node, and following the arc corresponding to the attribute value. For example, given an instance with $A_1 = True$ and $A_2 = False$ the decision tree in Figure 4.2 would give the classification $True$ by following the $True$ branch at node A_1 and the $False$ branch at node A_2 (the tree actually represents the boolean operator $A_1 \oplus A_2$). Decision trees are thus evaluated by performing a sequence of tests on the problem attributes where the order of attributes tested depend on the tree structure.

In our case, we have a set of discrete feature-value pairs represented by the agents described in the previous chapter (see Table 3.1 for an overview). Hence, when creating an *Agent Decision Tree* each agent instance is added as an internal node in the tree with outgoing arcs corresponding to the agent's output values. For example, the node associated with the Volume Agent has two outgoing arcs labeled Strong-Volume and Weak-Volume. Our target function has three output values resulting in three classification types; buy, sell or hold, which are added as leaf nodes in the tree. Figure 4.3 shows an example agent decision tree with six agent instances. Agent decision trees can be created with any number of agent instances.

As explained previously, decision trees are evaluated by traversing the tree from root to classification by testing the feature at each internal node and following the arc corresponding to the feature value. Similarly, agent decision trees are evaluated by polling the agent associated with each internal node for its output signal and following the arc corresponding to the output value. For example, given the example tree in Figure 4.3, if the Trend Agent reports that the stock is in a downtrend we continue by polling the Volume Agent. Then, if the Volume Agent detects that the stock is trading with

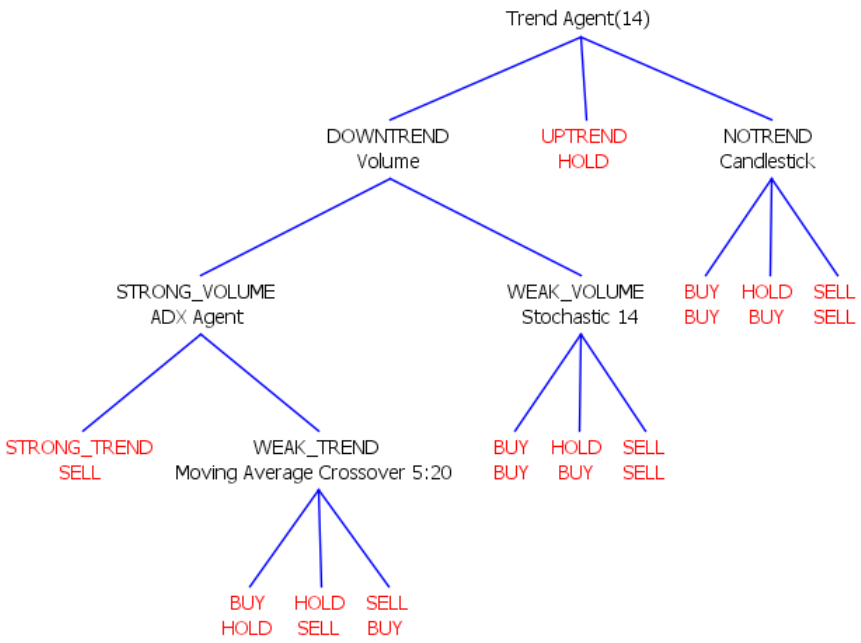


Figure 4.3: An example agent decision tree. Arc labels are placed over the agent name and classification nodes are colored red.

relatively low volume and the Stochastic Agent finds that the stock is overbought we end up with a sell classification. Thus, an agent decision tree represents an an investment decision reached by performing a sequence of tests on different technical indicators, where the sequence and type of indicator tested depend on the tree structure. This is the same approach used by many technical traders, at least according to a selection of the technical analysis literature (Murphy, 1999; Turner, 2007). Decision trees thus seem like an appropriate representation for the target function. The algorithm developed for learning agent decision trees is explained in the following section.

4.3 Evolving Decision Trees

In the general decision tree classification framework, a classification is reached by performing a sequence of tests on the problem attributes. Thus, the output classification depend on the tree structure and the arrangement of classification nodes. Decision Tree Learning consequently involves searching for a tree structure that adequately classifies the training data. Most DTL algorithms stem from the ID3 algorithm and its successor C4.5. All traditional DTL algorithms are designed for supervised learning and thus require labeled training data. Traditional DTL algorithms are consequently not applicable in our case of unsupervised learning. As a result, an Agent Decision Tree Learning (ADTL) algorithm was created. The algorithm is an *evolutionary algorithm* that evolves an agent decision tree for a particular stock through a process of artificial evolution.

4.3.1 Artificial Evolution

Evolutionary algorithms are a class of optimization and search algorithms that are inspired by concepts in biological evolution such as inheritance, natural selection, mutation and mating. The sophisticated, complicated, robust and adaptive features observed in biological systems serve as a powerful motivator for mimicking these processes in problem-solving systems. Evolutionary algorithms are often used for problems that are hard to solve using traditional optimization techniques such as optimizing discontinues functions and/or functions with many non-linearly related parameters (Floreano and Mattiussi, 2008). They have been applied with successful results to a whole range of practical and scientific problems, for example, finding novel and patentable analog and electrical circuit designs (Koza et al., 2003), evolving antenna designs to be used on a NASA aircraft (Lohn et al., 2005), and more traditional benchmark problems such as the Travelling Salesman Problem (Floreano and Mattiussi, 2008). Due to elements of randomness and a highly parallel search, the details of which will be described shortly, evolutionary algorithms perform a much wider search of the problem space compared to traditional optimization algorithms. As a result, evolutionary algorithms often find novel solutions that could not have been obtained using traditional techniques.

Progress in biological evolution require the presence of the following four factors, each of which are present to some extent in an evolutionary algorithm (Floreano and Mattiussi, 2008).

1. A population of two or more individuals.
2. Diversity in the population (the individuals are not identical).
3. Heredity (individual traits can be transmitted to offspring through reproduction).
4. Selection (only part of the population is allowed to reproduce).

In evolutionary biology the population goes through a cycle of development (i.e., growing up), natural selection by surviving and being able to reproduce in some environment, and death. Diversity in the population is maintained as the result of reproduction when offspring are created as small variations of their parents. Reproduction also results in heredity as each parent transmits some of their genetic material (i.e., the "blueprint" of the organism, such as DNA in biological organisms) to the offspring. The genetic material of an organism is known as the *genotype* while its manifestation as an organism is known as the *phenotype*. Natural selection then ensures that only the strongest individuals are able to reproduce by favoring certain individual traits over others (i.e., survival of the fittest). Thus, inheritance works solely on the genotype, while natural selection works solely on the phenotype. As this process of recombination, heredity and natural selection continues over successive generations, the population gradually evolves and adapts to its environment.

This same process, including concepts of a genotype, phenotype, population, diversity, heredity and selection is modeled in most applications of evolutionary algorithms. Thus, individuals in evolutionary algorithms are typically represented by a genotype and a phenotype that can be derived from the genotype. Following the analogy from biological evolution, the genotype is often some simple low-level data structure while the phenotype

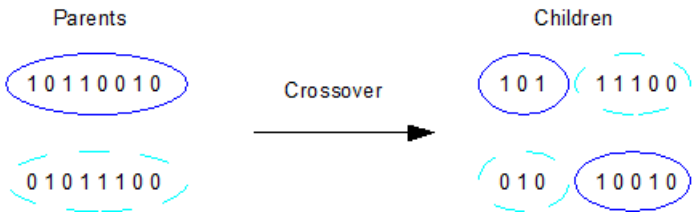


Figure 4.4: Binary crossover.

represents a candidate solution that can be executed on the optimization problem.

Different types of evolutionary algorithms are often distinguished by the representation of the genotype. For example, in genetic algorithms the genotype is represented by a simple bit string. A development procedure then needs to be specified so that each bit string can be translated to a candidate solution. For example, a genetic algorithm designed to solve the Travelling Salesman Problem (TSP) may encode a candidate solution as a sequence of cities to visit where each city is denoted by an integer. The development procedure can then be specified as a simple binary to integer conversion. In other cases, the genetic encoding is more elaborate, such as in Genetic Programming where each genotype represents a potential computer program.

The genotype representation needs to be designed so that it allows for heredity in reproduction. Hence, it must be possible to define a process that splits and recombines different genotypes into new genotypes that can still be translated into valid phenotypes. In evolutionary algorithms this process is typically implemented by a set of genetic operators. The genetic operators often include a crossover operator that combine parts of two parent genotypes to create a new child genotype, and a mutation operator that creates small variations in a single genotype. The crossover operator is thus primarily responsible for heredity (although it also introduces some diversity), while the mutation operator is responsible for diversity. In genetic algorithms the mutation operator is often defined as a simple bit flip in the genotype bit string. An illustration of crossover in a bit string genotype is given in Figure 4.4. In some evolutionary algorithms the reproduction operators are defined directly on the phenotype (if the phenotype can be logically separated and recombined) and no separate genotype encoding is used.

Natural selection in evolutionary algorithms are modeled by a fitness function and a selection strategy. The fitness function measures how well the phenotype of an individual (i.e., candidate solution) solves the given optimization problem. As such, the fitness function along with the phenotype representation are the aspects of the algorithm that define the problem to be optimized. For example, a possible fitness function for the TSP problem could return the length of the tour represented by the input phenotype. Once the fitness value for each individual has been obtained, natural selection can be executed by using a selection strategy that performs a weighted stochastic selection process based on the fitness values. Different selection strategies implemented will be discussed in detail in Section 4.3.5. The general cycle of an artificial evolutionary cycle is illustrated in Figure 4.5.

4.3.2 Evolutionary Cycle

The evolutionary cycle begins in the lower left corner of Figure 4.5. It is initiated with a population of a fixed number of randomly created genotypes. The population is then passed to the development phase where each genotype is translated into a phenotype representing a candidate solution. The population is then sent to the fitness test phase where the fitness function is executed on every phenotype. The quality of each phenotype is thus evaluated by executing the phenotype on the given problem, resulting in a fitness value being associated with each individual. Next, two selection strategies are executed on the population. First, a number of individuals are chosen to be adults based on some criteria. The individuals not chosen to be adults are typically removed from the population (i.e., they die), making room for new offspring. Secondly, the adults compete among themselves, based on their fitness, to be chosen as parents. Once two parents have been selected, the reproduction phase is entered where the genetic operators are executed on the parent genotypes to create new child genotypes. This process of parent selection and recombination is continued until the population is back to its original size. The cycle then starts again with the new population, and typically continues for a number of generations. Once the evolutionary cycle ends (typically after a fixed number of generations or when the population has reached a predefined fitness threshold), the fittest individual in the population is chosen as a solution to the target problem.

The rationale behind the process is the same as in evolutionary biology. As the selection strategies are designed to stochastically choose phenotypes with a bias towards high fitness, and the recombination operators are designed to maintain some heredity, successively higher fitness phenotypes are produced. That is, poor fitness phenotypes (poor candidate solutions) are less likely to be chosen for reproduction and will eventually be removed from the population (they die) while high fitness phenotypes (good candidate solutions) are allowed to reproduce and hopefully carry on their good traits to the next generation. Stochasticity is important in the selection strategies as poor fitness genotypes may hold good traits that have not been complemented with other traits to show their utility and high fitness genotypes may represent local optima. We thus want to allow some poor genotypes to continue to evolve.

In the following sections each element of the evolutionary cycle as implemented in the Agent Decision Tree Learning (ADTL) algorithm will be explained.

4.3.3 Genotype and Phenotype Representation

As our goal with the ADTL algorithm is to find an agent decision tree that maximizes potential profits, the phenotypes (i.e., candidate solutions) are decision trees as described in Section 4.2. The genotype representation is typically implemented as a simple linear representation of the phenotype that facilitates the reproduction operators (i.e., creating new genotypes by separating and combining parent genotypes). However, as reproduction operators are easily specified on tree structures by separating and recombining sub-trees, the ADTL algorithm employs a direct mapping between genotype and phenotype. That is, the ADTL algorithm operates directly on a population of decision trees, there is no conversion from a lower-level genotype. This eliminates the need for an expensive

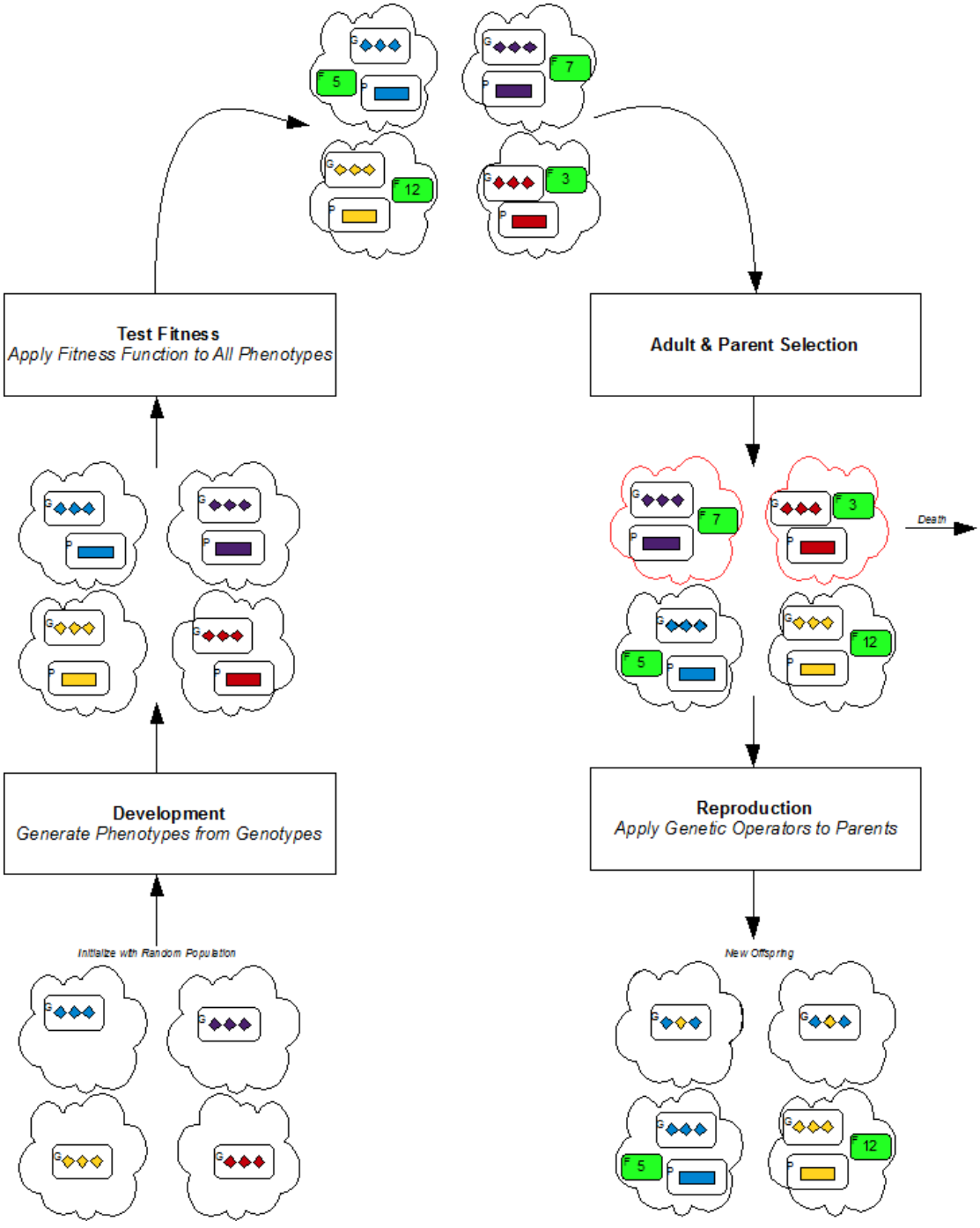


Figure 4.5: Basic cycle in artificial evolution.

development procedure for translating genotypes to phenotypes. This is the same approach used in genetic programming where each phenotype is an executable computer program organized in a tree structure where internal nodes are operator functions and terminal nodes are operands (Lisp is a programming language that can be structured as a tree).

The classification nodes are added as leafs with a bias to its connected arc. That is, a buy classification will be added to a buy arc with some probability (say 70%), with sell and hold being equally probable. For example, when the Candlestick Agent outputs buy it is more likely to be assigned to a buy classification because the domain knowledge implemented by the agent suggests that this is a good opportunity to buy. For agents that do not output buy, sell or hold, we have no prior knowledge of the appropriate classification so classification nodes are added with no bias.

4.3.4 Fitness Testing

The fitness function should give objective and graded evaluations of the phenotypes. Graded evaluations are essential as the selection strategies and consequently progress in the evolutionary cycle will stagnate if the entire population is relatively equal in fitness. Moreover, the fitness function represents what we want our phenotypes to accomplish (i.e., the function that we want to maximize). As a result, the fitness function is defined according to the performance measure described in Section 4.1. That is, the fitness function performs a portfolio simulation based on the investment strategy represented by a phenotype (i.e., an agent decision tree) over the training data. In the fitness assessment stage the fitness function is executed on every phenotype in the population, making it the most computationally expensive phase of the evolutionary cycle.

The portfolio simulation is initialized with some amount of investment capital and subsequently evaluates the input decision tree for every day in the training data. Simulated transactions are then executed based on the classification produced by the decision tree for each day. As the investment strategy represented by a phenotype is associated to a single stock, we purchase as many shares as possible using the current available investment capital on every buy signal and sell all shares currently owned of the stock on every sell signal. We thus have either all money placed in the stock or no money placed in the stock. The fitness value returned by the function is the profit-ratio obtained at the end of the simulation (i.e., the initial investment capital divided by the total value of the portfolio at the end of the simulation). The actual amount of initial investment capital is thus irrelevant as we measure the increase or decrease related to this amount. Extended details on the design and implementation of the portfolio simulation procedure are described further in Section 5.2.

4.3.5 Selection Strategies

Selection strategies in evolutionary algorithms are essential for the population to evolve. The purpose of the selection strategies is to introduce some stochasticity into the process of survival and mating. This is done by performing weighted stochastic selection with a bias towards high fitness.

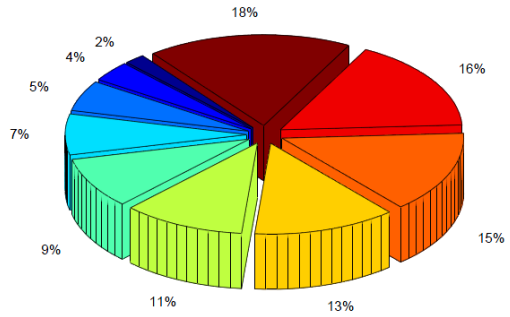
The ADTL algorithm includes two levels of selection, adult selection followed by parent selection. In adult selection, a number of individuals are selected as candidates for reproduction. Three different strategies have been implemented for adult selection,

1. Full Generational Replacement
2. Over-Production
3. Generational Mixing

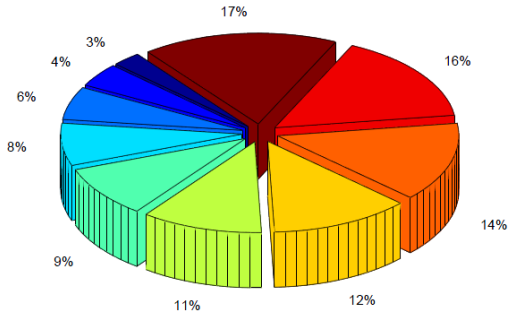
In full generational replacement all adults from the previous generation are removed on every pass through the evolutionary cycle. All offspring generated by reproduction thus gain free entrance to the adult pool, resulting in no selection pressure on offspring. In over-production all adults from the previous generation are similarly removed, but there is a maximum limit on the number of adults allowed in the population which is smaller than the number of offspring generated. Thus, the offspring must compete among themselves for the adult spots using the same mechanism as in parent selection. For example, given a population of 60 individuals with a fixed adult pool of size 20, the 20 adults will be removed on each generational pass, and the 40 children will compete, based on their fitness, for the 20 adult spots. Naturally, 40 new offspring will, in this case, need to be produced on each pass through the cycle. In generational mixing the entire population compete for a spot in the adult pool. In this case selection pressure on offspring is very high as they are competing with some of the best individuals that have evolves so far.

Weighted stochastic selection in evolutionary algorithms is often implemented by *roulette wheel selection*. In roulette wheel selection each individual is assigned to a roulette wheel where the space occupied by each individual is proportional to the individual's fitness. Selecting two parents for reproduction is then done by spinning the wheel twice. The sector space allocated to each individual is determined by normalizing and stacking the fitness values. For example, given a population of 4 individuals (denoted a,b,c,d) with the following fitness values, [a:4, b:3: c:2, d:1], the fitness values are first normalized by dividing by the total sum of fitness, 10, and stacked so that each individual gets a sub-range in the interval [0, 1). The roulette wheel is thus divided in four sectors such that a:[0, 0.4), b:[0.4, 0.7), c:[0.7, 0.9), d:[0.9, 1.0). Spinning the roulette wheel then simply involves generating a random number in the range [0,1) and selecting the individual that occupies the sector pointed to by the generated number. Figure 4.6 illustrates a roulette wheel with 10 individuals.

In many cases, the individual fitness values are scaled prior to normalization. The above example with normalized but unscaled fitness values is called fitness proportionate selection. Scaling fitness values is done to control the selection pressure. For example, given a population of individuals with one individual significantly dominating in fitness, the high fitness individual will receive a very large slot on the roulette wheel and consequently dominate the selection process which may lead the population to prematurely converge to the high fitness individual, an example of which can be view in Figure 4.7a. One scaling approach implemented in ADTL called *sigma scaling* uses the population's fitness variance as a scaling factor,



(a) Roulette wheel with normalized but unscaled fitness values.

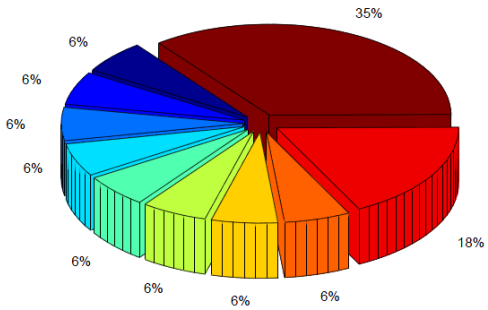


(b) Roulette wheel with sigma scaled and normalized fitness values.

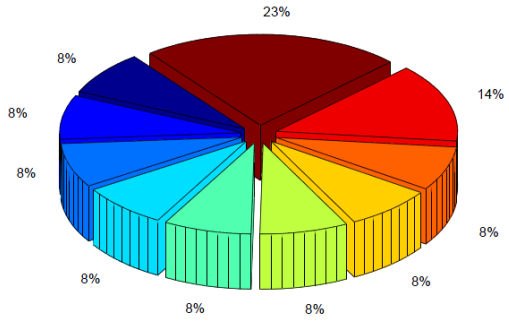
Figure 4.6: The original fitness values are $1, 2, 3, 4, 5, 6, 7, 8, 9, 10$. Figure adapted from (Downing, 2009).

$$Scale(i, g) = 1 + \frac{f(i) - \bar{f(g)}}{2\sigma(g)} \tag{4.3}$$

where g is the generation number, $f(i)$ is the fitness of individual i, $\bar{f(g)}$ is the population's fitness average in generation g, and $\sigma(g)$ is the standard deviation of population fitness. Sigma scaling effectively alters the selection pressure in cases when a few individuals are much better than the rest of the population (as the standard deviation of population fitness, $\sigma(g)$, would then be high). It also helps to increase selection pressure in cases where the population has very similar fitness (i.e., low $\sigma(g)$). Figure 4.6 and 4.7 show two examples of sigma scaling compared to fitness proportionate scaling.



(a) Roulette wheel with normalized but unscaled fitness values. The high fitness individual is alloted a large sector on the roulette wheel.



(b) Roulette wheel with sigma scaled fitness values. Notice the more even distribution.

Figure 4.7: The original fitness values are $1, 1, 1, 1, 1, 1, 1, 1, 3, 6$. Figure adapted from (Downing, 2009).

4.3.6 Reproduction

The purpose of the reproduction operators is to provide variation in the population so that progress in the evolution does not stagnate and to allow the favorable traits of one generation to be passed on to the next. The ADTL algorithm includes two reproduction operators, crossover and mutation.

The mutation operator does a simple random operation to re-structure a candidate tree. When applied to a tree the operator re-builds the tree by selecting nodes at random from the original tree, creating a new tree likely very different from the original tree. As such, the mutation operator serves the purpose of diversifying the population. In this way, the algorithm always has some chance to escape from potential local optima. The operator is associated with a mutation rate that dictates how often an individual will be chosen for mutation (e.g., there is a 5% probability that any given phenotype is chosen for mutation). Like the other parameters associated with ADTL, the mutation rate needs to be manually tuned. If set too aggressive, the algorithm will loose direction and progress in the search, resulting in a population of random individuals. If set too low, the algorithm might converge to a local optima early in the run, also resulting in a lack of progress.

The crossover operator executes a random subtree swapping between two candidate trees. It takes two parent trees, t_1 and t_2, and returns two new child trees, c_1 and c_2. The crossover is performed by first selecting a random *split node* in each tree, s_1 and s_2. Each split node then defines a subtree in their respective parent. The subtrees, as defined by s_1 and s_2, are then swapped such that the parent of s_1 becomes the new parent of s_2, and vice versa. If s_1 and s_2 are selected such that they are both the root in their respective trees (s_1 is the root in t_1 and s_2 is the root in t_2), no crossover is performed and c_1 and c_2 are returned as identical copies of their parents. The crossover operator is illustrated in Figure 4.8. Here the Candlestick Agent was chosen as s_1 in the first parent tree and the ADX Agent was chosen as s_2 in the second parent tree. The result of the swap, shown in the bottom half of the figure, are two agent decision trees fairly different from their parents.

Once the crossover has been performed, the new child trees are pruned to ensure that no agent is used more than once along each branch. The pruning mechanism operates by traversing each branch and keeping a record of the agents evaluated along each path down the tree. If an agent is encountered twice along the same path, the latter agent is replaced by the sub-tree connected to the arc associated with the latter agent output. The purpose of the pruning mechanism is to decrease memory usage and increase computational efficiency, it has no effect on the classifications generated by a candidate tree.

Depending on the random choice of the split node, the offspring created by the crossover operator might be very diverse or very similar to their parents. As mentioned above, if the root node is chosen to split in both parent trees, the children returned are identical copies of their parents. If, however, an internal node is chosen in both trees, two very diverse children might be created. Moreover, if two leaf nodes are chosen to split in both trees, the children will be very similar, but not identical to their parents. s_1 and s_2 can also be chosen as to be classification nodes, making a simple swap of classification. This was essentially a design goal when creating the crossover operator as a good balance

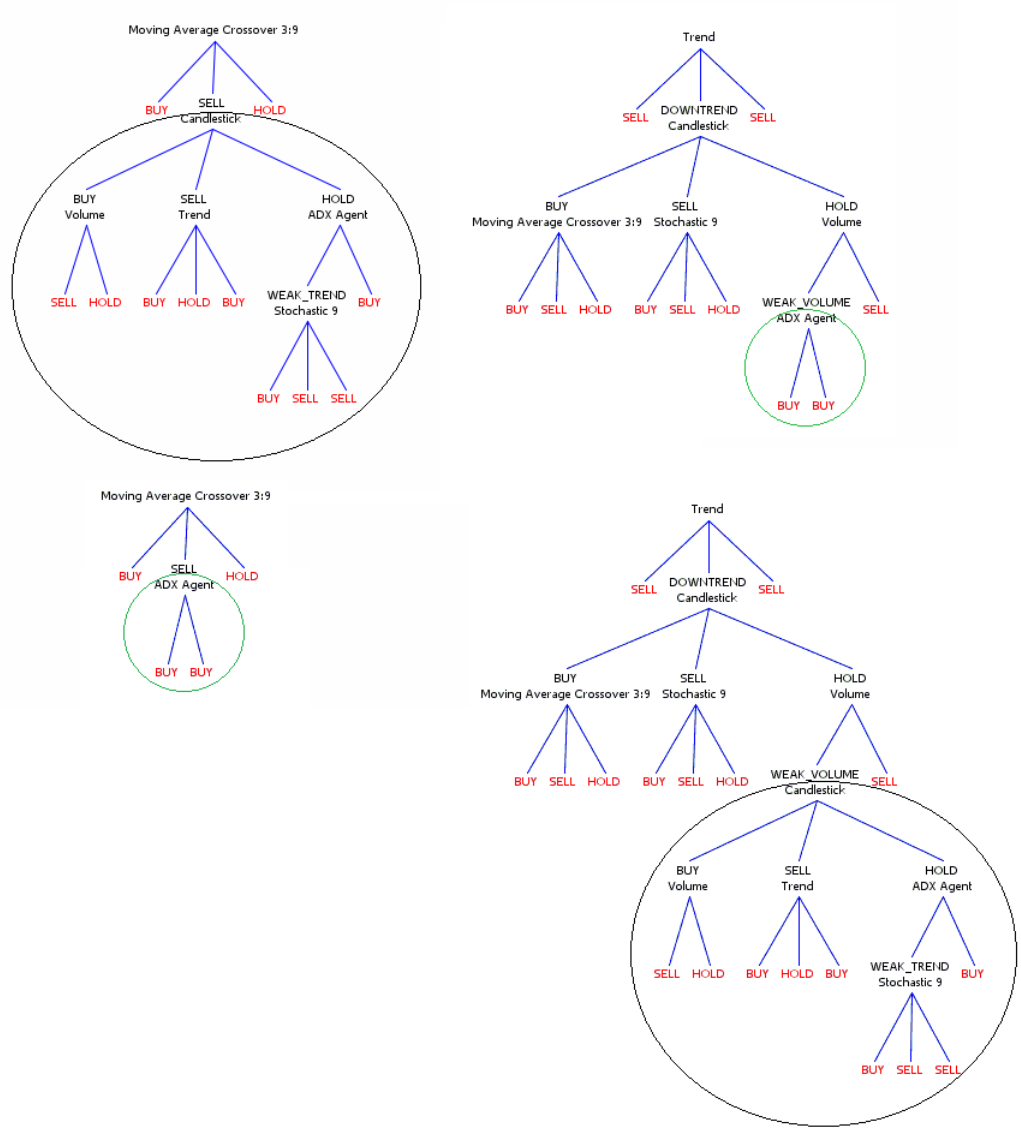


Figure 4.8: Agent Decision Tree Crossover

between heredity and randomness improves the direction and progress of the search while simultaneously protecting against getting stuck in local optima (balance exploitation with exploration).

4.3.7 ADTL Parameters

As noted in (Floreano and Mattiussi, 2008), one major drawback with evolutionary algorithms is the number of parameters that have to manually tuned for each application. The parameters associated to the ADTL algorithm are given in the left column of Table 4.1. Through extensive empirical tests during development we have found that the values given below allow us to evolve sufficiently high fitness phenotypes within reasonable time. On the development laptop (a Dell XPS M1330 with a Intel Core2 Duo 2.1GHZ CPU and 2GB RAM running on Windows 7) evolving a decision tree for a single stock over 200 generations with a population size of 100 decision trees takes approximately 15 to 20 seconds. Evolving a tree for every stock in the dataset (53 in total) then takes approximately 15 minutes.

Parameter	Value
Population Size	100
Adult Pool Size	50
Generations	200
Elitism Rate	1
Mutation Rate	0.15
Adult Selection	Generational Mixing
Parent Selection	Roulette-Wheel with Sigma-scaling

Table 4.1: ADTL Parameters

5 Portfolio and Money Management

The *Portfolio and Money Management* module is responsible for two tasks; running portfolio simulations and deciding how much money to invest on a given trade signal. Portfolio simulations are used as the fitness function in the ADTL algorithm and to evaluate the performance of the entire system.

5.1 Money Management

The money management strategy is responsible for determining the amount of money to invest on a given trade signal. A simple approach to this problem would be to always invest a fixed fraction of available investment capital, a fixed sum, or a fixed number of shares on every trade signal. However, as all trade signals are generated by evaluating price data, we want to utilize the price data to calculate a confidence measure associated to each trade signal (e.g., we are 60% confident that a buy signal generated for Norwegian Air Shuttle (NAS) will result in positive returns). Essentially, more money should be invested on trade signals that have been profitable in the past, according to some criteria. Trade signals that are, historically, less profitable should be awarded less money so that more money is available for historically profitable signals. Moreover, trade signals with a history of negative returns may be disregarded. Naturally, there is no guarantee that historical success can be extended to the future, but we assume that some money management strategy based on a measure of historical success is better than the naive approach of always investing a fixed quantity. This assumption will be evaluated and discussed in Chapter 6.

The strategy used by the money management module is based on the *Kelly Criterion* (Kelly, 1956), an optimal betting strategy for favorable gambling games. Although its application was originally described for gambling games, it has also been applied to the stock market and portfolio management with positive results (Browne and Whitt, 1996; Rotando and Thorp, 1992). The Kelly Criterion is based on maximizing a quantity G defined as the exponential growth rate of a gambler's capital,

$$G = \lim_{N \to \infty} \frac{1}{N} \log \frac{V_N}{V_0}$$

where V_0 is the gambler's initial capital and V_N is the capital after N bets. Consider a bet that we know we can win with a probability $p > 0.5$ with b-to-1 odds. We have 100 NOK to wager and we are given even money odds (i.e., $b = 1$, we double up or loose the entire bet). If we decide to bet a fraction $f = 0.2$ of current money we get $V_1 = (1 + bf)V_0 = (1 + 0.2)100 = 120$ if we win or $V_1 = (1 - f)V_0 = (1 - 0.2)100 = 80$

if we loose. In general, given that V_0 is the initial capital and V_N is the capital after N bets, we have $V_N = (1 + bf)^W (1 - f)^L V_0$ where W and L are the number of wins and losses in the N bets. The Kelly Criterion then identifies a fraction f of the current bankroll to bet that will maximize the exponential growth rate of V_N,

$$G(f) = \lim_{N \to \infty} \frac{W}{N} \log(1 + bf) + \frac{L}{N} \log(1 - f)$$

$$= p \log(1 + bf) + q \log(1 - f)$$

where $\lim_{N \to \infty} \frac{W}{N} = p$ is the probability of winning and $\lim_{N \to \infty} \frac{L}{N} = 1 - p = q$ is the probability of loosing. If we derive $G(f)$ and solve for f we find the Kelly Criterion,

$$f^* = \frac{bp - q}{b}$$

where f^* is the fraction of current capital to wager on each bet in order to maximize G in the long run. For example, given a 1-to-1 bet with a 60% probability of winning the Kelly Criterion says that the gambler should bet a fraction $\frac{(1 \times 0.6) - 0.4}{1} = 0.2$ of current money in order to maximize the exponential growth rate of the gambler's capital. If $b < q/p$ the Kelly value will be negative, thus indicating that the gambler should take the opposite side of the bet (if possible).

When applied to investing we need to calculate estimates for the odds b and probability of winning p. One approach considered is to interpret the b parameter as the possible gain received by investing in the stock. For example, if we purchase some stock and believe that its price will increase by 5% the gain is $g = 1.05$. If we keep g fixed, calculating the probability of "winning" the investment would simply involve counting the number of buy signals that reached the target gain within some maximum allowable time period (e.g., 10 days). For sell signals we interpret the g parameter as the potential gain received by *not* owning the stock. f^* is then interpreted as the fraction of the current volume of stock (if any) that should be sold. For example, lets say we have 10 buy signals generated over the training data for NAS (Norwegian Air Shuttle). If the price of NAS increased by the target gain (say 5%) within 10 days in 7 out of the 10 signals we get $p = 0.7$ for the buy signals generated over NAS. For sell signals we use a target decrease of the same magnitude to calculate p (i.e., we have essentially gained g by selling the stock). That is, if we have 10 sell signals where the price in 6 out of the 10 signals dropped by 5% within 10 days we get $p = 0.6$ for the sell signals generated over NAS.

For the agent decision trees described in Section 4.2 the probability p using the interpretation of the odds as a target increase/decrease may be calculated for every buy/sell classification, respectively. Thus, each classification reached by an agent decision tree is associated with a success rate that can be used in the Kelly formula to determine the number of shares to buy or sell given the classification. Figure 5.1 shows a randomly created decision tree for the Oslo Benchmark Index with success rates associated to each classification node calculated over the training data.

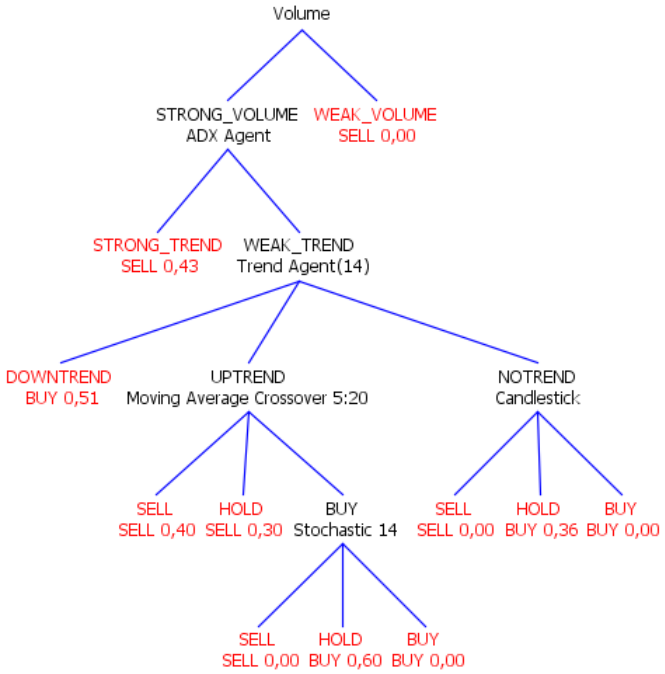


Figure 5.1: Agent Decision Tree with success rates calculated for each classification node.

As b needs to be stated in b-to-1 odds, we need to use $b = g - 1$ in the Kelly formula. If we use $b = g - 1$ in the formula and assume that g remain fixed we find that $p > \frac{1}{g}$ to obtain $f^* > 0$. Thus, if we use a target increase of 5% we get $p > \frac{1}{1.05} = 0.95$ or with a target increase of 10% we get $p > 0.91$. It may seem implausible that over 95% of the classifications will reach the target gain of 5%. We consequently also implement an alternative interpretation for the b-parameter.

Traders often interpret the odds b as the win/loss ratio associated to historical trades [1]. The win/loss ratio is defined as the average gain of positive trades divided by the average loss of negative trades. A positive trade is defined as a trade that generated positive returns while a negative trade is a trade that generated negative returns. In our case, a buy trade is defined as one or more buy signals followed by a sell signal. A sell trade is defined by one or more sell signals followed by a buy signal (i.e., a short trade). For example, lets say we have 3 buy trades, with two trades generating positive returns. For the two positive trades the stock price increased by 3 NOK and 5 NOK, respectively. For the negative trade, the stock price decreased by 4 NOK. We thus get an average gain for the positive trades of $(3 + 5)/2 = 4$ and an average loss for the negative trades of 4, giving $b = 4/4 = 1$. We then define the probability of winning a trade p as the number of winning trades divided by the total number of trades. Thus, we get $p = 2/3$, and for future buy trades we get a Kelly value of $\frac{(1\times(2/3))-(1/3)}{1} = 0.33$. We should thus invest 33% of current capital on future buy signals.

[1] `http://www.investopedia.com/articles/trading/04/091504.asp`

5.2 Portfolio Simulations

Portfolio simulations are executed by simulating buy and sell transactions based trade signals generated by some prediction model and an amount of initial investment capital (the default is 100 000 NOK),

$$\text{Portfolio-Simulation}(M, I, P_s) = \frac{\text{Portfolio-Value}_1}{\text{Portfolio-Value}_T}$$

where $M(P_s, t) \rightarrow \{\text{Buy, Sell, Hold}\}$ is a prediction model for $P_s(t)$, I is the initial investment capital, Portfolio-Value_1 and Portfolio-Value_T is the total value of the simulated portfolio at the start and end of the simulation, respectively. When initiated the simulation iterates over $t = [1, \cdots, T-1]$ (i.e., each day in the training or test data) and execute $M(P_s, t)$ to generate a trade signal. A transaction corresponding to the trade signal is then executed on day $t+1$. As we use daily price data, trade signals will only be generated at the end of each business day so the transaction has to be executed on the following day. If a buy signal is generated, we assume that we are able to buy the stock at the opening price on the following day, $P_s(t+1)$. The same holds for sell signals, we assume that the stock can be sold at the opening price the following day. In reality, these assumptions may not necessarily hold as we may not be able to find a buyer/seller willing to fill our order at the opening price.

The amount of stock to buy or sell is either defined by a fixed order total or determined by the money management strategy. For example, if the simulation is executed with a fixed order total of 1000 NOK we purchase shares for 1000 NOK on every buy signal (i.e., the number of shares to buy is $\lfloor 1000/P_s(t+1) \rfloor$), and sell all shares (if any) on every sell signal. In some portfolio simulations we use an order total equal to the current available investment capital. That is, we either have all money invested in a single stock or all money as free capital. In this case, consecutive buy signal will be disregarded as there is no money to invest after the first buy signal. When using the money management strategy, the strategy calculates a fraction f^* of current available capital and the number of shares to buy is given by $\lfloor (\text{capital}_t \times f^*)/P_s(t+1) \rfloor$.

For example, given a buy signal for Norwegian Air Shuttle (NAS) on 10-05-2010, we assume that the stock can be bought at the opening price on 11-05-2010, the number of shares to purchase is calculated by the money management module or a fixed order total is used, and given that there is sufficient capital to buy, the stock is added to the portfolio and the free investment capital is reduced according to the cost of purchasing the shares. If there is no available capital to purchase the shares the trade signal is simply ignored. When sell signals are processed the number of shares sold is subtracted from the current portfolio and the profits are credited to the free investment capital. If a sell signal is generated for a stock that we do not currently own the signal is simply ignored (i.e., we do not handle short selling).

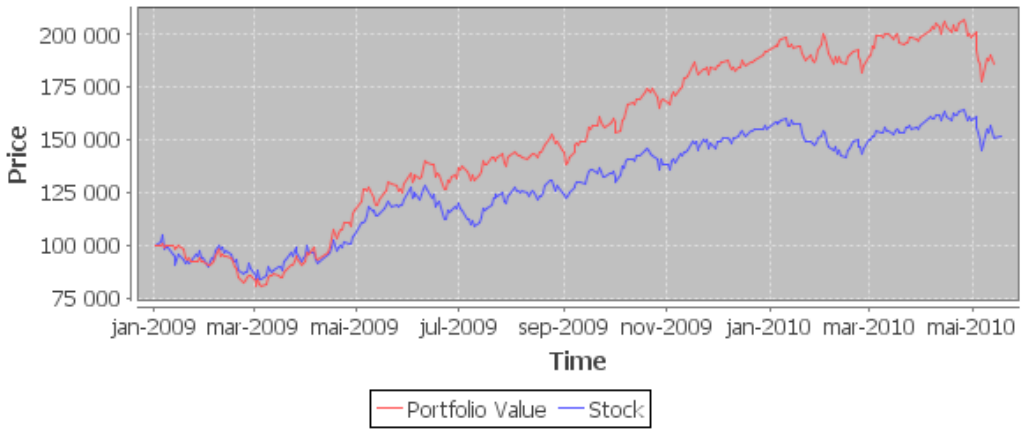
Portfolio simulations can also be executed for a set of stocks (rather than a single stock as described above), which is more realistic compared to real trading. In that case, the portfolio simulation takes a set of stocks $S = [s_1, \cdots, s_n]$ and for each s executes $M(P_s, t)$

to generate a trade signal for stock s at time t. The result is a set of trade signals generated for different stocks each day. If the money management strategy is used the values generated by the strategy are employed to prioritize the trade signals. We thus interpret the money management value as a measure of signal strength and prioritize signals with a high money management value. If we want to invest more money in the stock, we should be more confident in the signal, thus we choose the signal with the strongest investment value. Transactions are then processed according to the trade signals until a maximum transaction limit is met. The transaction limit is important as we potentially have a buy or sell signal for every stock in the dataset. The transaction limit is typically set to one buy and one sell transaction each day. If the money management strategy is not used we execute every generated trade signal (given that there is enough free investment capital).

For example, using the money management strategy and a transaction limit as defined above; given three signals on 10-05-2010, including two buy signals; one for Norwegian Air Shuttle (NAS) and one for Seadrill (SDRL), and a sell signal for Blom (BLO), the signals are first sorted according to their money management value. If buy signals for NAS have been more successful in the past than buy signals for SDRL, a buy transaction for NAS is executed and the trade signal for SDRL is ignored. The sell signal for BLO will be executed regardless as there was no competing sell signals.

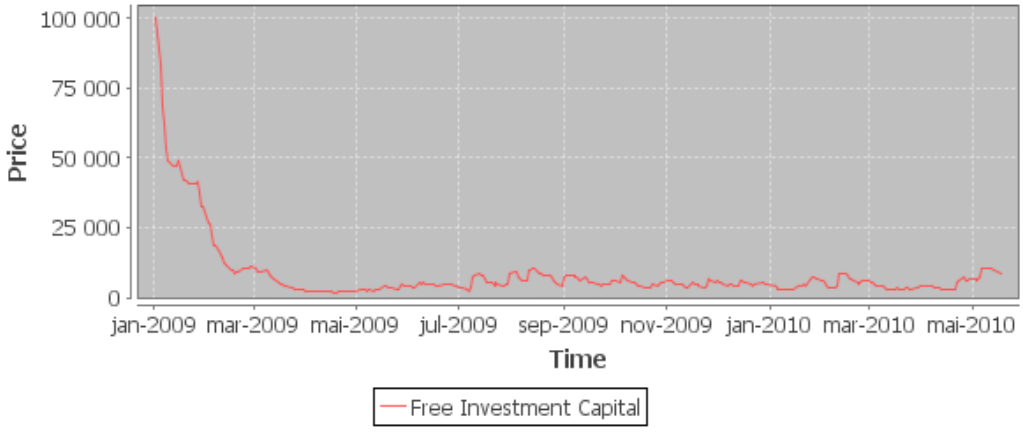
In addition to calculating the profit-ratio at the end of a simulation, portfolio simulations also generate plots of the portfolio value history, portfolio value distribution, free investment capital history and transaction history (using a Java library called JFreeChart[1]). The portfolio value history is the total value of the simulated portfolio at any point in time. That is, for any point in time the portfolio value is calculated as the accumulated value of the number of shares owned in each stock multiplied by its closing price, plus any free investment capital. For example, on 10-05-2010 a simulated portfolio may contain 100 shares of NAS, SDRL and BLOM. If we assume that the closing price for each stock on 10-05-2010 is 10 NOK and we have 1000 NOK in free investment capital, we get a portfolio value of $10 \times 100 + 10 \times 100 + 10 \times 100 + 1000 = 4000$. Portfolio value history plots can also be generated for a single stock, in which case we assume that the entire investment capital is placed in the stock at the start of the simulation. The portfolio distribution shows the value of each stock in the portfolio in a bar chart. The free investment capital plot simply shows the amount of investment capital not tied up in stocks at each point in time. Some sample plots are given in Figure 5.1,

[1] `http://www.jfree.org/jfreechart/`



(c) Portfolio Value History

(d) Portfolio Value Distribution



(e) Free Investment Capital History

Figure 5.1: Plots generated by portfolio simulations.

6 Results and Discussion

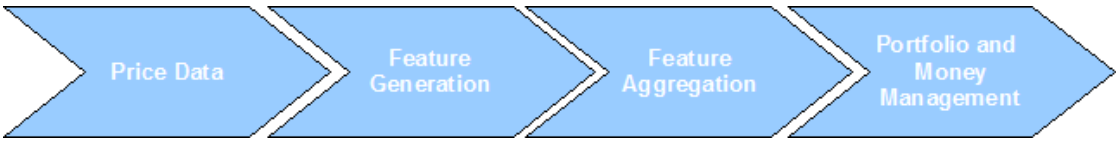


Figure 6.1: The Stock Price Prediction Model

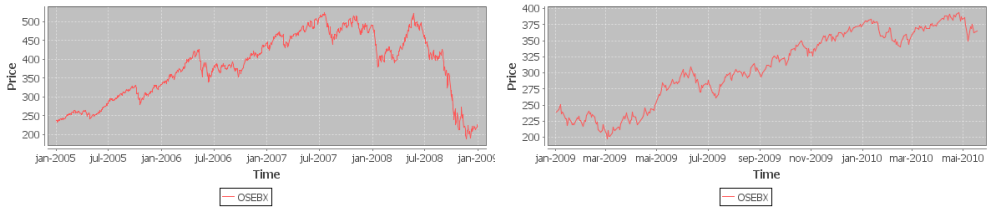
This chapter presents the results obtained by testing the developed stock price prediction model (Figure 6.1) described over the past three chapters on price data from the Oslo Stock Exchange. Returning to the research question and success criteria initially stated in Section 1.4, we want to evaluate if the developed prediction model is capable of selecting stocks to trade with performance comparable to a professional technical trader. The primary measure of performance is thus based on running portfolio simulations using predicted trade signals generated for all stocks in the dataset described in Section 1.3. Portfolio simulations provide a quantitative basis for analysis through profit-ratios and other derived data, and a qualitative basis for discussion through generated transaction histories.

As each reasoning layer (i.e., Feature Generation and Feature Aggregation) may serve as independent prediction models each layer will be evaluated individually. Consequently, in Section 6.1 we evaluate the Feature Generation module in isolation, in Section 6.2 we evaluate the results of using machine learning to aggregate features in the Feature Aggregation module, and finally, in Section 6.3 we evaluate the utility of the money management strategy and the performance of the entire model applied as an artificial trader.

In order to remain consistent in evaluation and discussion the same experiment procedure is executed on both reasoning layers. That is, when evaluating the Feature Generation and Feature Aggregation module portfolio simulations will be run using the same parameter settings. In order to evaluate the full utility of each module we allow the portfolio simulation to execute every generated trade signal (i.e., there is no transaction limit) and we use a fixed order total of 1000 NOK for every buy signal. That is, for every generated buy signal 1000 NOK is invested in the associated stock and for every generated sell signal we sell all shares currently in the simulated portfolio. We initialize portfolio simulations with a high initial investment capital (500 000 NOK) to ensure that the simulation never runs out of money. The amount of initial investment capital is not really relevant as long as the simulation always have sufficient funds to execute the predicted trade signals.

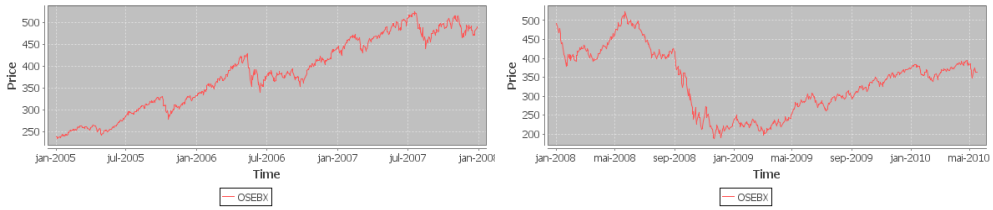
The dataset described in Section 1.3 is divided in two training and test sets to evaluate different problem scenarios. In the first scenario (denoted Dataset A) the model is trained with data from 01-01-2005 to 31-12-2008 and tested with the remaining data from 01-01-2009 (see Figure 6.2). In this scenario we evaluate how well the model performs in an essentially *good* year where the benchmark index is in an uptrend. In the second

scenario (denoted Dataset B) the model is tested with data from 01-01-2008 (see Figure 6.3). In this scenario we evaluate how well the model performs during a period of high recession due to the financial crisis. We thus asses if the model is capable to regain potential losses as the recession turns in the start of 2009.



(a) Training Data (b) Test Data

Figure 6.2: Dataset A - A "Good" Year



(a) Training Data (b) Test Data

Figure 6.3: Dataset B - A "Difficult" Year

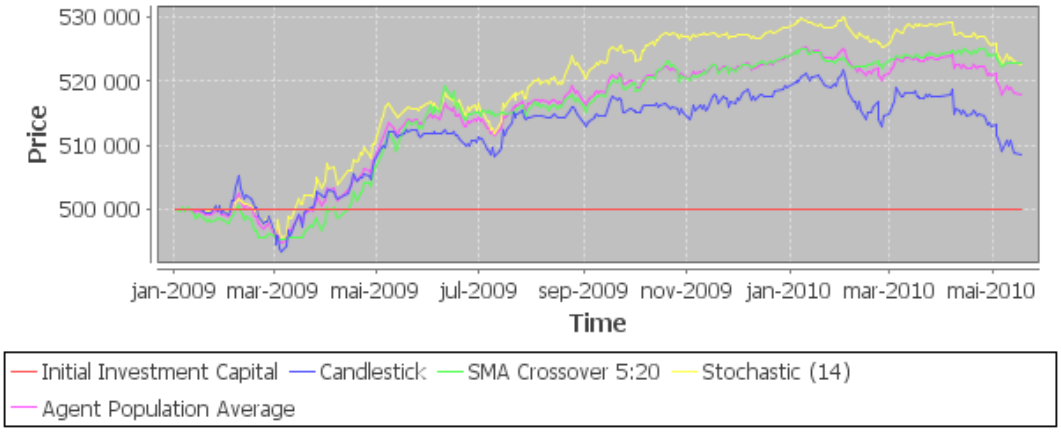
6.1 Feature Generation Results

The Feature Generation module contains a population of agents that implement domain knowledge from technical stock analysis. That the agents correctly implement their intended domain knowledge was empirically validated to some extent in Chapter 3 and in more detail in (Larsen, 2009). The focus here is thus to evaluate the performance of the technical indicators implemented by the agents in terms of profits generated over portfolio simulations. Naturally, as agents that implement measures of trend, trend strength and volume do not generate trade signals (i.e., buy, sell and hold), they are difficult to evaluate in terms of profitability alone. A discussion of these agents is thus delayed to Section 6.2 where their utility in combination with the other agents will be discussed. The agents evaluated here are thus agents that implement measures of trend reversal and thereby generate buy and sell signals.

Each agent instance is tested by running it through the portfolio simulation described in Section 5.2. That is, for each day in the test data the portfolio simulation gathers trade signals generated by the input model (in this case an agent instance) for all stocks in the dataset. Simulated transactions are then executed based on the collected trade signals. The portfolio parameters used were outlined in the introduction to this chapter (i.e., we

use an initial investment capital of 500 000 NOK and a fixed order total of 1000 NOK). We use the initial investment capital as a baseline measure of performance. Thus, if an agent instances generates trade signals that yield profits above the initial investment capital we consider the agent successful.

As some agents are parametrized, we execute the experiment twice for two sets of agent instances with different parameters. First, the Candlestick Agent, SMA Crossover Agent and Stochastic Agent are tested with their default parameter settings. Next, we evaluate two different parameter settings for the SMA Crossover Agent and Stochastic Agent, thereby evaluating the effect of using different parameters in terms of generated profits.



(a) Portfolio value history for the first set of agent instances over Dataset A.

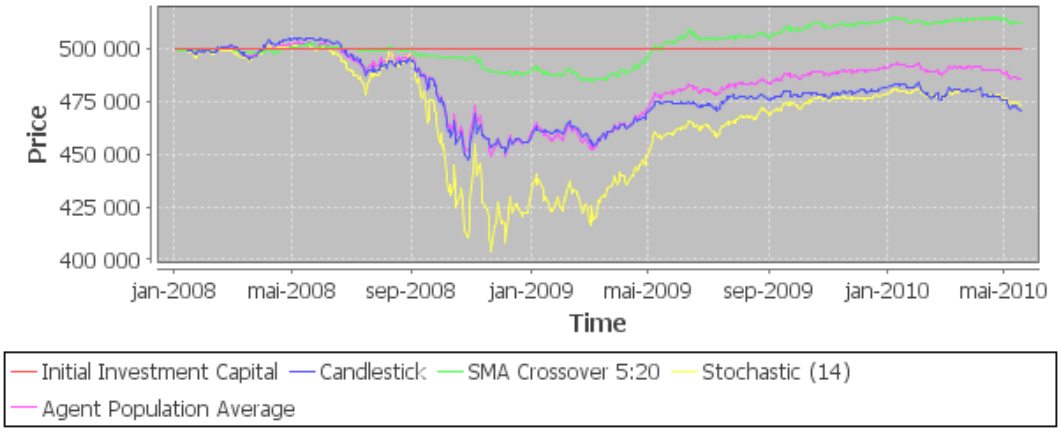
Agent	Buy Signals	Sell Signals	Portfolio-Value	Profit
Candlestick	724	283	508518	8518
SMA Crossover (5:20)	577	575	522800	22800
Stochastic (14)	216	72	522485	22485
Average	506	310	517942	17942

(b) Transaction history overview.

Figure 6.4: Results for the fist set of agent instances over Dataset A.

Figure 6.4 show the results obtained by executing the experiment on the first set of agent instances over Dataset A. The results show that every agent instance generate positive returns with the SMA Crossover Agent producing the best end-result with a profit of 22800 NOK. The Stochastic Agent generates the best overall results considering the portfolio value history. On average, the three agent instances generates a profit of 17942 NOK. If we compare the number of transaction executed by each agent against the profits obtained it does indicate that a large number of spurious signals are generated. However, as every agent instance does yield an end-result with positive returns, the profitable signals do seem to outweigh the unprofitable signals. In this respect, the Stochastic Agent is again the best-performing agent as it executes far fewer transactions than the other two agent instances, thus indicating that it generates a higher ratio of profitable signals. The poor performance of the Candlestick Agent may be because it is designed to detect short-term trend reversals and in the experiment stocks are only sold if a corresponding

sell signal is generated.



(a) Portfolio value history for the first set of agent instances over Dataset B.

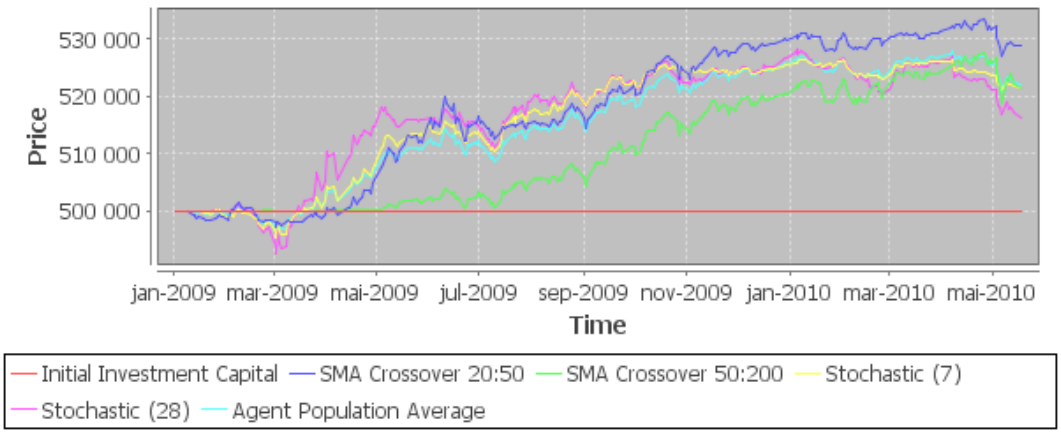
Agent	Buy Signals	Sell Signals	Portfolio-Value	Profit
Candlestick	1313	513	471100	-28900
SMA Crossover (5:20)	954	952	512758	12758
Stochastic (14)	588	143	473579	-26421
Average	952	536	485821	-14179

(b) Transaction history overview.

Figure 6.5: Results for the fist set of agent instances over Dataset B.

Figure 6.5 show the results obtained by executing the experiment on the same set of agent instances over Dataset B. Here, the performance of the agents is considerably worse with the SMA Crossover Agent being the only agent yielding a profit. This is not a surprising result considering that the financial crisis hit in mid-2008 which resulted in a dramatic price fall for every one of the 53 stocks in the dataset. On average, stock prices fell by approximately 60% from May 2008 to early January 2009, making it a period of time when it was practically impossible to generate profits from trading. To obtain a profitable result in this period a trader would have to a) sell all shares in the portfolio before prices started to decline, or b) execute a large number of smart short-term trades catching small upturns in price.

In this respect, the performance of the SMA Crossover Agent in this period is quite significant. If we examine the transaction history generated by the agent we find that the it predominantly generates sell signals in the period from May 2008 to October 2008, resulting in a small portfolio (i.e., few stocks) during the worst period of price decline. The domain knowledge implemented by the agent thus seem to work as expected; once price starts to fall the shorter moving average falls below the longer moving average and a sell signal is generated. The domain knowledge implemented by the Stochastic Agent and Candlestick Agent seem to be less applicable in this downtrend period as they continue generating a large number of buy signals throughout 2008. This is a result we hope the Feature Aggregation will be capable to learn.

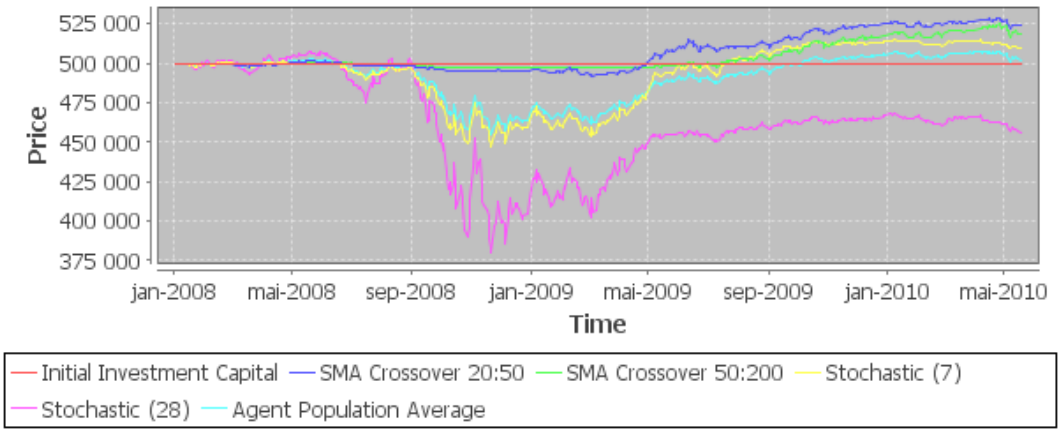


(a) Portfolio value history for the second set of agent instances over Dataset A.

Agent	Buy Signals	Sell Signals	Portfolio-Value	Profit
SMA Crossover (20:50)	208	187	528899	28899
SMA Crossover (50:200)	56	16	521574	21574
Stochastic (7)	121	48	521969	21969
Stochastic (28)	237	61	516285	16285
Average	156	78	521970	21970

(b) Transaction history overview.

Figure 6.6: Results for the second set of agent instances over Dataset A.



(a) Portfolio value history for the second set of agent instances over Dataset B.

Figure 6.7: Continued on next page...

Agent	Buy Signals	Sell Signals	Portfolio-Value	Profit
SMA Crossover (20:50)	331	308	523927	23927
SMA Crossover (50:200)	73	33	518540	18540
Stochastic (7)	335	109	510198	10198
Stochastic (28)	755	127	456272	-43728
Average	374	144	502023	2023

(b) Transaction history overview.

Figure 6.7: Results for the second set of agent instances over Dataset B.

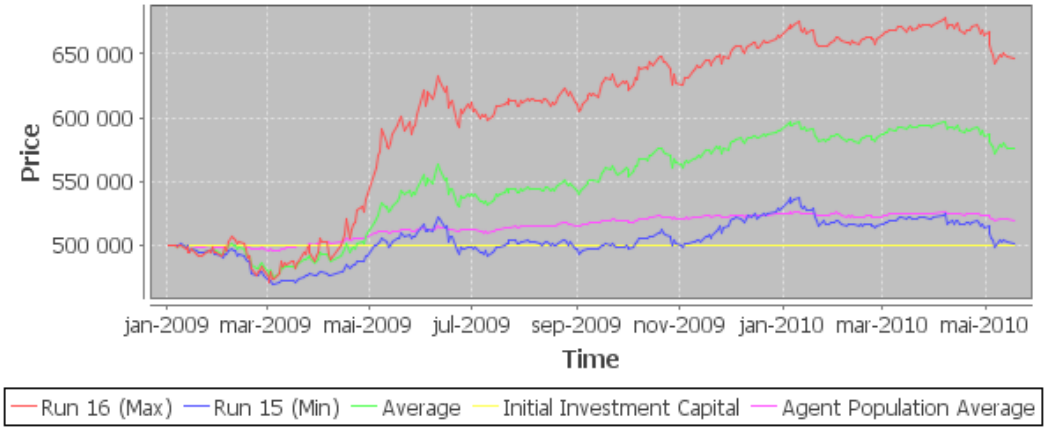
Figure 6.6 and 6.7 show the results obtained by executing the same experiment over the second set of agent instances over Dataset A and B, respectively. Over Dataset A, the results are fairly consistent with the above results with the second set of agent instances generating approximately the same profits as the first set of agent instances. Over Dataset B, however, we find that the second set of agent instances does generate better results than the first set of agent instances. This indicates that it may be useful to include some internal machine learning in the Feature Generation layer that learns appropriate parameters for each parametrized agent. This concept will be further explored in Chapter 7 where we discuss possible points for future work.

6.2 Feature Aggregation Results

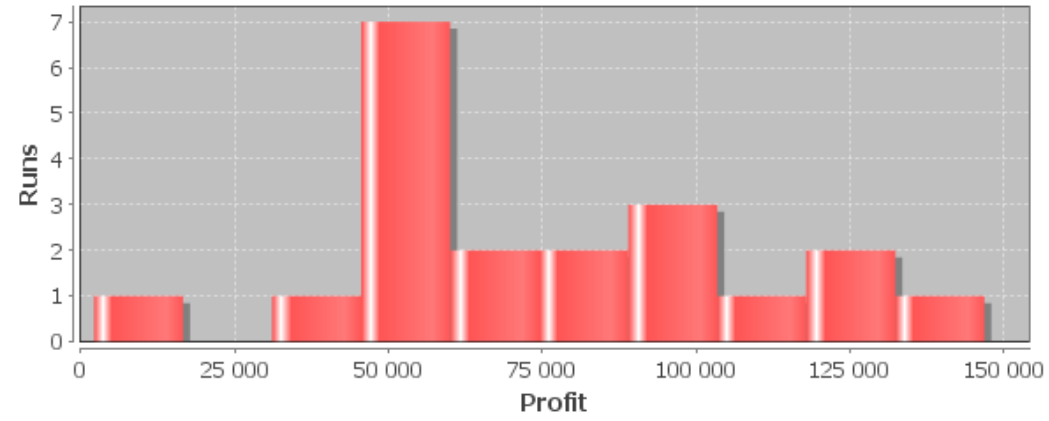
The purpose of the Feature Aggregation module is to combine and contextualize the features generated in the Feature Generation module. This is done by organizing agent instances in decision trees and searching for profitable decision trees using a developed evolutionary algorithm called the Agent Decision Tree Learning (ADTL) algorithm. In this section we evaluate the performance of using evolved agent decision trees as investment strategies compared to using the agents directly as discussed in the previous section.

The Feature Aggregation module is evaluated using the same experiment procedure outlined in the previous section. That is, the ADTL algorithm is executed on every stock in the dataset, creating 53 separate agent decision trees, one for each stock. The agent instances used include every instance evaluated in the previous section as well as the Trend Agent, ADX Agent and Volume Agent. For each day in the test data the portfolio simulation evaluates each decision tree, thus generating a trade signal for every stock in the dataset. The baseline measure of performance in this experiment is the portfolio value history generated by the agents. That is, if the Feature Aggregation module generates higher profits than the agents on which it is based, we consider the experiment a success. The parameters used in the ADTL algorithm was given in Section 4.3.7.

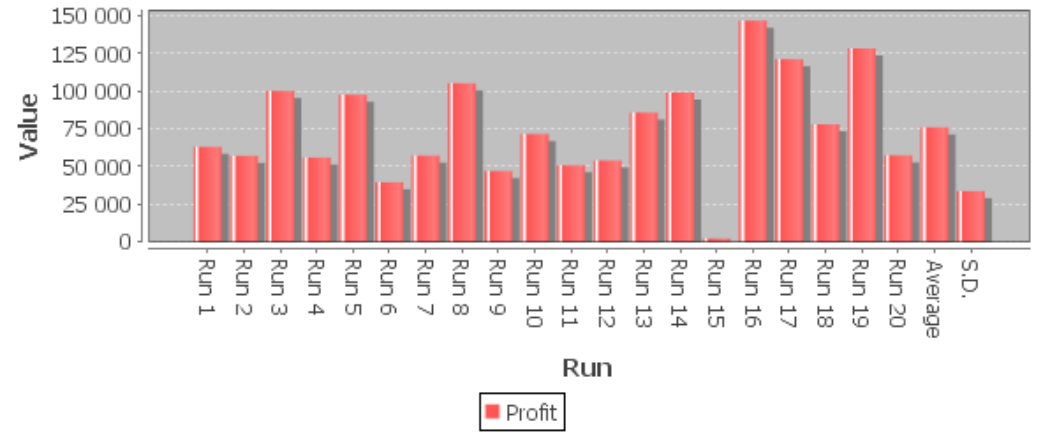
The results obtained by executing the experiment procedure over Dataset A are given in Figure 6.8. As the Feature Aggregation module is an inherently stochastic process, 20 runs of the experiment procedure are executed. The results show that the Feature



(a) Portfolio value history for the best, worst and average run.



(b) Histogram of profits generated by the 20 runs.



(c) Bar chart of profits generated by the 20 runs.

Figure 6.8: Feature Aggregation results over Dataset A. On average, portfolio simulations over the Feature Aggregation module execute 2767 transactions with 2203 buy transactions and 564 sell transactions.

Aggregation module generally outperform all agent instances tested in the previous section. The best run reached a profit of 146898, thus beating the best performing agents by approximately 400%. The poorest run, however, generated a profit of 2126, which is in line with the poorest performing agents. Averaged over the 20 runs, the Feature Aggregation model generates profits of 75 000, approximately 275% better than the agent population average. 7 of the 20 runs generated profits above 100 000.

Looking at the transaction history generated by each run we find that, on average, portfolio simulations over the Feature Aggregation module execute 2767 transactions where 2203 are buy transactions and 564 are sell transactions. As there is a total of 343 days in the test data, on average 6 buy transactions and 2 sell transactions are executed each day. Considering the high number of transactions executed, analyzing the transaction history generated by each run becomes a difficult task. Moreover, due to the inherent stochasticity in the module (each run is, in principle, based on a different set of 53 agent decision trees), it is difficult to determine the factor that distinguishes good runs from poor runs. We may thus question if the good performance of the module is the result of a higher number of transactions or a selected set of more profitable transactions.

In order to analyze this to some extent we continue by analyzing a single run of the ADTL algorithm for one of the 53 stocks. We thus leave a deeper analysis of the complete transaction histories generated by the module to Section 6.3 where we use a transaction limit making it more feasible to qualitatively analyze the transaction history.

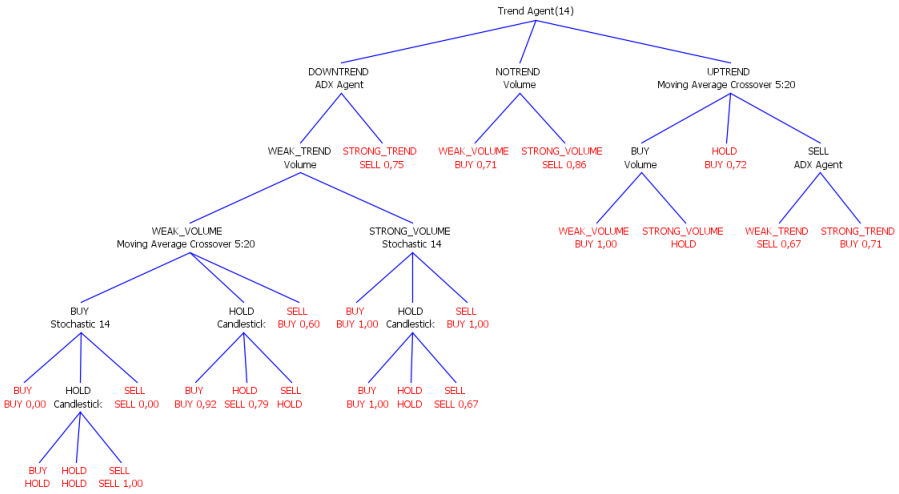


Figure 6.9: Evolved agent decision tree for Norwegian Air Shuttle (NAS) including success rates associated to each classification (see Section 6.3). The success rates are calculated using a target increase/decrease of 5% over a maximum of 10 days.

Figure 6.9 shows an agent decision tree for Norwegian Air Shuttle (NAS) evolved over the training data shown in Figure 6.11a. From Figure 6.10 we see that the average fitness of the population and the fittest individual gradually evolves over successive generations, eventually reaching a maximum fitness of 15.86. The ADTL algorithm thus works as

expected by generating successively higher fitness decision trees. Figure 6.11 show the portfolio value history generated by the fittest individual produced at the end of the evolutionary cycle over the training data.s The most encouraging observation from Figure 6.11a and 6.11b is that the investment strategy represented by the fittest decision tree successfully catches the uptrend at the start of the period and sells when the downtrend starts to form at the end of the period. The decision tree has thus successfully generated very profitable entry and exit points in the stock.

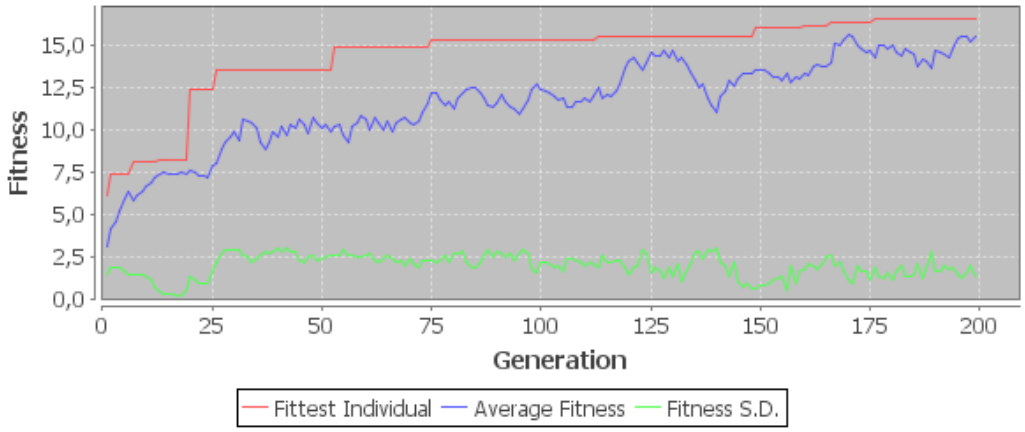
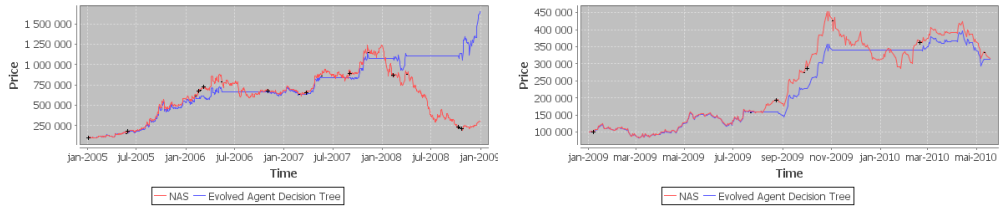


Figure 6.10: Fitness Progression for ADTL on Norwegian Air Shuttle (NAS)

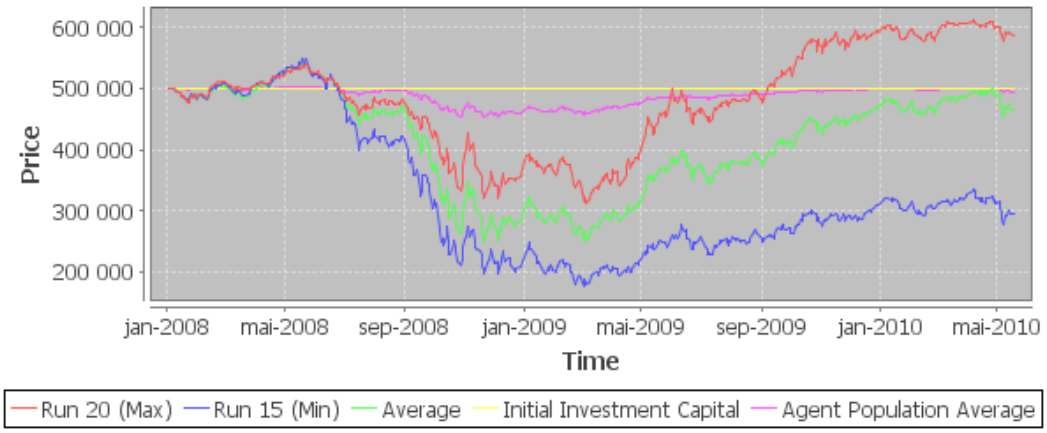


(a) NAS Portfolio Run Training

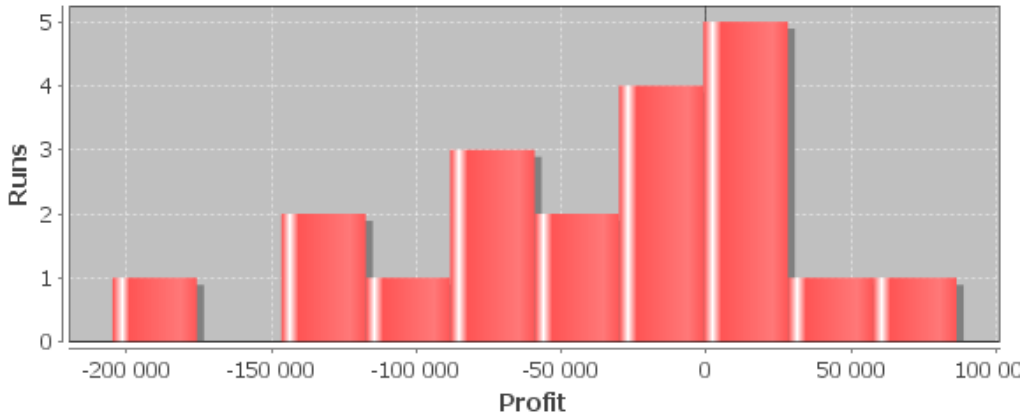
(b) NAS Portfolio Run Test

Figure 6.11: Portfolio value plot for the decision tree evolved for NAS (Norwegian Air Shuttle). Buy and sell signals are denoted by '+'/'-', respectively.

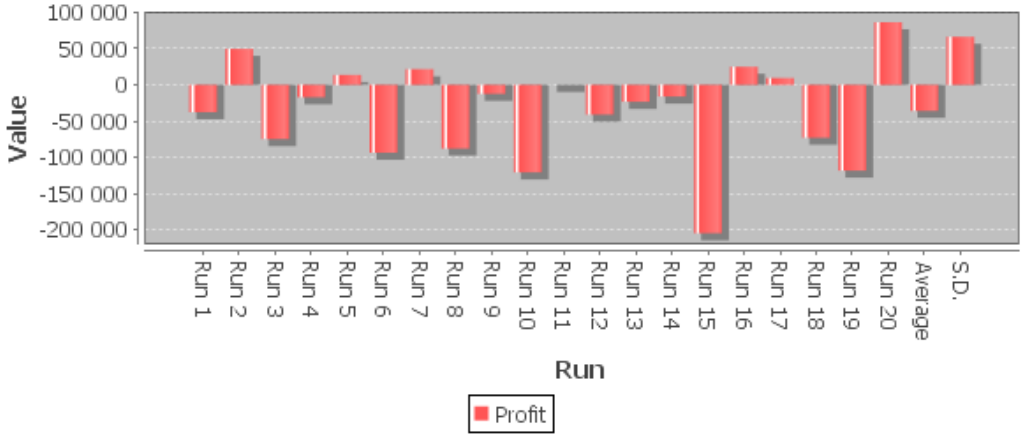
Doing a full analysis of the entire tree given in Figure 6.9 would be a challenging task considering its size and complexity. However, we can see that the tree does captures some important details about the training data. For one, the evolved tree has the Trend Agent placed at the root of the tree. As the training data begins in a strong uptrend and ends in a strong downtrend it seems reasonable to test the Trend Agent first. Furthermore, following the downtrend branch from the Trend Agent we reach the ADX Agent that measures trend strength. If the ADX Agent detects that the data is in a strong trend a sell signal is produced. Thus, the tree successfully captures the trading rule, "sell given a strong downtrend". The utility of this rule can be observed during the latter part of Figure 6.11a and 6.11b where sell signals are generated when the downtrend starts to form. If we follow the uptrend branch from the Trend Agent we find the Moving Average Crossover (MAC) Agent. Here, a buy signal is produced if the agent does not detect a crossover. This also seems reasonable given that we know the price is in an uptrend.



(a) Portfolio value history for the best, worst and average run.



(b) Histogram of profits generated by the 20 runs.



(c) Bar chart of profits generated by the 20 runs.

Figure 6.12: Feature Aggregation results over Dataset B. On average, 2927 transactions are executed with 2235 buy transactions and 692 sell transactions.

If the price is in an uptrend and we detect no moving average crossover (i.e., no trend reversal signal) it seems reasonable to buy.

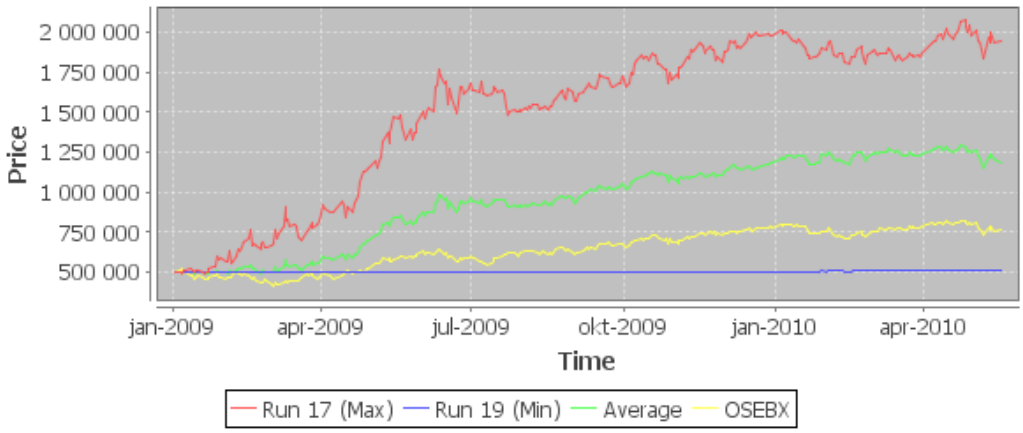
Given the above discussion it does seem like the ADTL algorithm is capable of learning some important aspects of the training data. Unfortunately, the results for Dataset B, given in Figure 6.12, are less encouraging. Although the best run generated a profit of 86 000, the average result is a loss of approximately 40 000. Moreover, 15 of the 20 runs result in no profits or negative returns. By examining the transaction histories generated we find that the module keeps making a large number of trades in the downtrend period of 2008. On average, 4 buy transactions and 1 sell transaction is executed each day in the test data. Thus, the Feature Aggregation module does not outperform the agent population in this experiment. This is not really a surprising result if we consider the training data used in this dataset. The training data used include data from January 2005 to January 2008 which is, in general, in a strong uptrend. The module has thus been trained in a period where it is very beneficial to purchase. In the following section we will go through a deeper analysis of the transaction histories generated in this period using the money management strategy.

6.3 Money Management Results

The money management strategy was implemented to determine the amount of money to invest on a given trade signal and to assess the strength of generated trade signals. In this section we evaluate the utility of the money management strategy applied to trade signals generated by the Feature Aggregation module (we will later refer to the Feature Aggregation module with and without the money management strategy as FA+MM and FA-MM, respectively). This section thus documents the final results with the complete prediction model employed as an artificial trader.

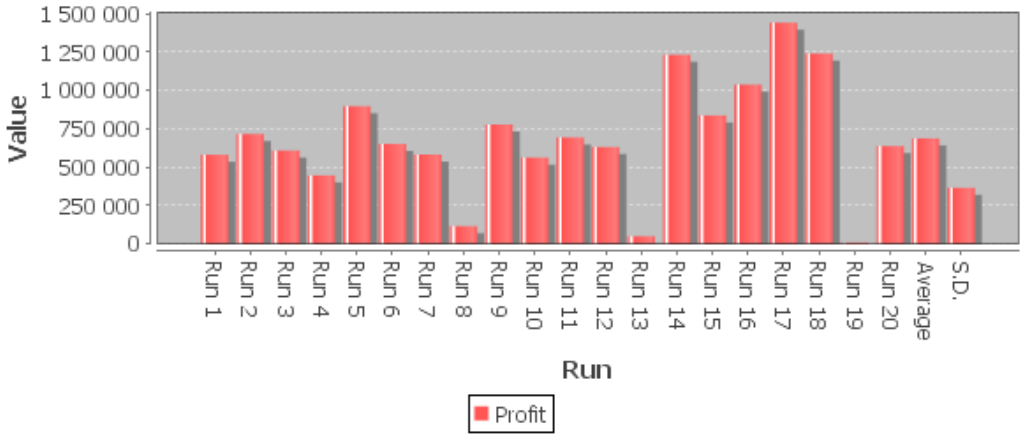
The experiment is executed with the money management strategy employed to determine the amount of stocks to buy and sell during training and testing. We thus use a different fitness function for the ADTL algorithm; rather than investing the entire investment capital on each buy signal and selling all shares on each sell signal, the fitness function will employ the same portfolio simulation as used by the experiment procedure. During training, the money management module incrementally adapt the probability measures used by the strategy. During testing, the money management values remain fixed. The money management values are thus trained indirectly by the ADTL algorithm.

The portfolio simulations are executed using the same parameter settings as used when evaluating FA-MA except that we use the money management strategy to determine the amount of shares to purchase and sell rather than investing a fixed amount. As the results from the two previous sections show that a large number of trade signals are typically generated, we impose a transaction limit of 1 buy and 1 sell transaction each day. The money management value associated to each trade signal is then used to select between competing signals (i.e., a trade signals with high money management values will be prioritized). The money management strategy thus acts as a signal filter for each day in the training and test data. The baseline measure of performance in this



(a) Portfolio value history for the best, worst and average run.

(b) Histogram of profits generated by the 20 runs.

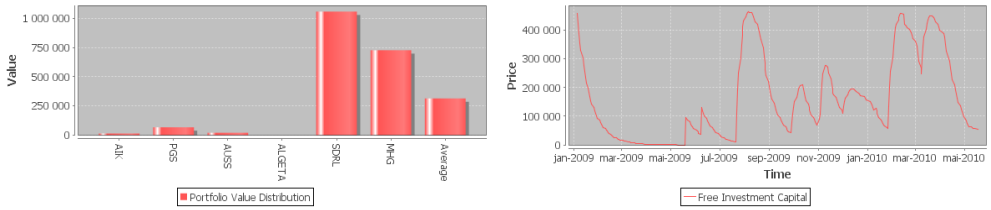


(c) Bar chart of profits generated by the 20 runs.

Figure 6.13: FA+MA results over Dataset A.

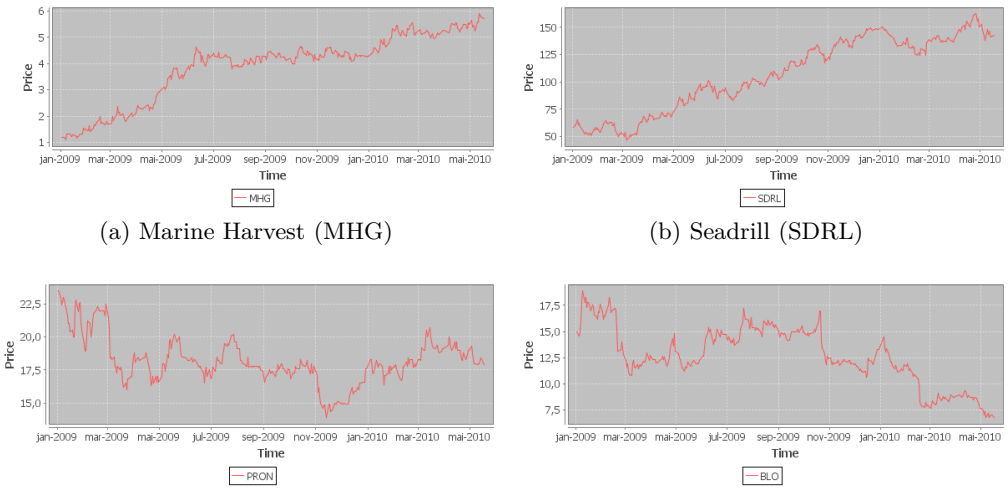
experiment are the results from the previous section (i.e., FA-MM). We will also compare the results against placing the entire investment capital in the Oslo Benchmark Index (OSEBX). That is, if FA+MM generate better results than FA-MM and higher profits than investing the entire capital in the benchmark index we consider the model a success. Placing 500 000 NOK in OSEBX would result in a profit of 261 565 over Dataset A (an increase of 52%) and a loss of 129 243 over Dataset B (a decrease of 25%).

Figure 6.13 show the results generated by the experiment on FA+MM over Dataset A. Although Figure 6.13c indicate that there are larger variations in the performance of each run, the average profit over the 20 runs is 630 775, over doubling the amount of initial investment capital. Compared to the results in the previous section, the average profits for FA+MM are 740% better than the average profits for FA-MM and 141% better than placing the entire capital in OSEBX. We also find that 15 of the 20 runs manage to double the initial investment capital. The best run almost tripled the initial capital with a profit of 1 439 501.



(a) Portfolio value distribution (b) Free investment capital history

Figure 6.14: Data from run 17.



(a) Marine Harvest (MHG)

(b) Seadrill (SDRL)

(c) Pronova BioPharma (PRON)

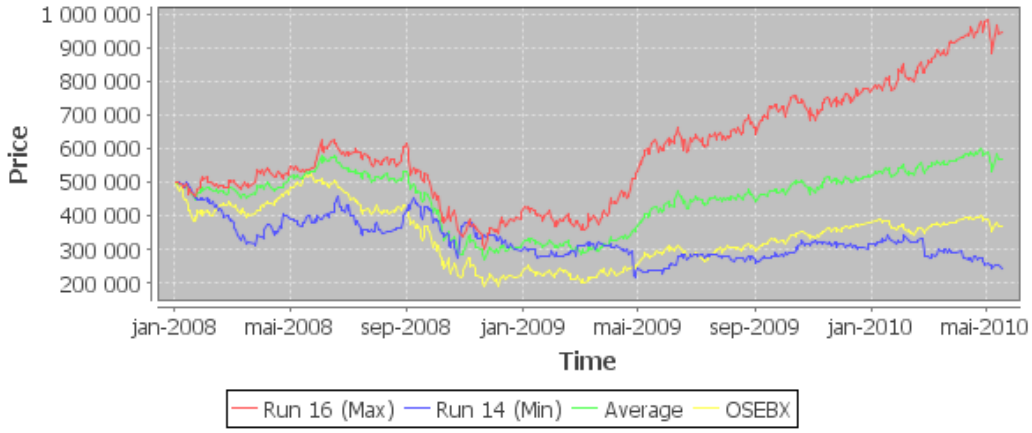
(d) Blom (BLO)

Figure 6.15: Price history for two profitable stocks (MHG and SDRL) and two less profitable stocks (PRON and BLO).

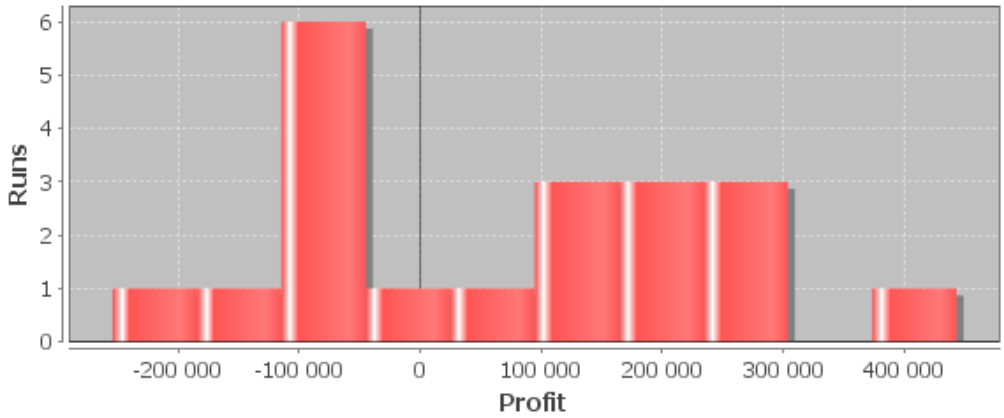
If we continue by analyzing the portfolio value distribution at the end of the best run, given in Figure 6.14a, we see that the distribution is dominated by two stocks, Marine

Harvest (MHG) and Seadrill (SDRL). Furthermore, by looking at the transaction history we find that the first transactions executed in the run are large buy orders for MHG. This behavior is further reflected in a plot of the free investment capital history generated by the run, given in 6.14b, where the amount of investment capital not tied up in stocks decreases rapidly from the start of the simulation to July 2009 where a large sell order for MHG is executed. The model then subsequently starts to invest in Seadrill and re-purchase some shares in MHG. If we examine the price history for MHG and SDRL in Figure 6.15 we find that they are both in a strong uptrend during the period, making them very profitable to hold. With an increase of 308% in 2009, MHG is the fifth best-performing stock on OSE in 2009. This result is consistent with other good runs as well, they are all dominated by a selection of a few well-performing (i.e., profitable) stocks, such as TGS, COP, ATEA, SCH and MHG. The poor runs, on the other hand, are dominated by a selection of a few poor performance stocks such as Pronova BioPharma (PRON) and Blom (BLO). It thus seems like the performance of the module is dominated by its ability to select a few well-performing stocks early in the portfolio simulation.

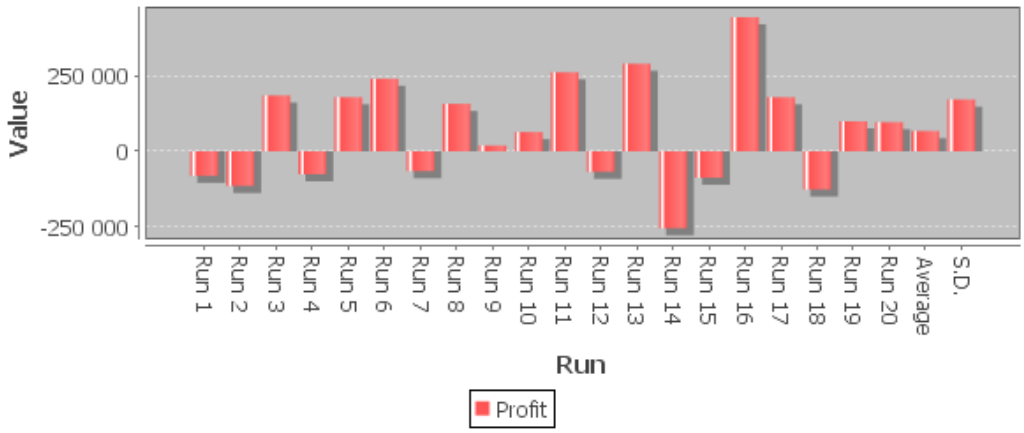
Figure 6.16 show the results obtained over Dataset B. Not surprisingly, the profits generated over this dataset are not as substantial as in Dataset A. However, on average the model successfully outperforms both the results from the previous section without the money management strategy and OSEBX. On average, the model generates a profit of 63340 NOK, a fairly good result considering the average return for FA-MM was a loss of 37650 and placing the entire capital in OSEBX would result in a loss of 129 243. Moreover, 12 runs generate a profit while 8 runs result in negative returns. The best runs from this experiment are fairly consistent with the best runs over Dataset A. That is, the model invests a large sum of money in a few profitable stocks. For example, in run 16 the module primarily invests in Royal Caribbean Cruises (RCL), Telenor (TEL) and Marine Harvest (MHG), all of which increase rapidly in price from the start of 2009. The model does not, however, refrain from trading during the period of high recession. As we have noted before, this may be a result of the training data in this dataset being in a strong uptrend. The model has consequently not been trained on a similar case before, which makes it particularly difficult to generate predictions for this period. In this respect, we may view the above results as successful.



(a) Portfolio value history for the best, worst and average run.



(b) Histogram of profits generated by the 20 runs.



(c) Bar chart of profits generated by the 20 runs.

Figure 6.16: FA+MA results over Dataset B.

7 Conclusion

The results documented in the previous chapter show that the developed prediction model using domain knowledge, machine learning and a money management strategy can create substantial profits when evaluated on the Oslo Stock Exchange. Some portfolio simulations increase the initial investment capital by almost 300% from January 2009 to May 2010, thus beating the Oslo Benchmark Index by approximately 250%. Although the model is less profitable when evaluated from January 2008, we do find that it does perform surprisingly well considering the global financial crisis that occured in mid-2008. In many simulations the model has regained the loss incurred by the financial crisis by mid-2009 and starts generating profits by late-2009, a significant result considering that the benchmark index fell by approximately 300 points from mid-2008 to 2009. We may thus conclude that the research question and success criteria stated in Section 1.4 have been successfully fulfilled, essentially to a greater extent than what was expected when the research question was first conceived and the success criteria defined.

The two-layer reasoning architecture is also deemed successful. For one, the agent-oriented design of the first reasoning layer allows for easy integration with new analysis techniques and adaptability by simply adding and/or removing agents from the agent population, facilitated by strictly defined agent and layer interfaces. As each layer may serve as independent prediction models, we get the added ability to evaluate each layer without interference from other parts of the system. The second layer of reasoning is also highly adaptable as the fitness function employed by the ADTL algorithm can be easy extended with additional constraints or other measures of success.

The omission of transaction costs in the portfolio simulation procedure is perhaps the most apparent flaw in the results documented in the previous chapter. Transaction costs can be difficult to model as they depend on the broker used, the order total of the executed transaction, tax deductions, and various other aspects. We have thus focused more on giving the prediction model elements of adaptability so that extensions, such as a transaction cost model, can be easily integrated in the existing system. Moreover, as the Feature Aggregation module learns its investment strategies from experience through a specific performance measure, it seems reasonable to think that it will also learn how to cope with transaction costs. In the final results, we employed as transaction limit which would reduce the effects on transaction costs on the results. A more thorough statistical analysis of the predictions generated by the different model modules would have been beneficial in validating the results. However, due to time constraints this was not completed in time and will consequently be added as a point for future work.

The most apparent problem with the prediction model is the inherent stochasticity in the model. While some portfolio simulations triple the initial investment capital, other simulation runs produce very little or no profits. Although few portfolio simulations have resulted in a loss of capital, it may seem like a challenging task to trust the model with real (rather than simulated) money. Moreover, this difference in profitability between

runs can be difficult to explain as each run is based on a different set of agent decision trees. As a result, we conclude this conclusion by outlining some possible points for future work that may be used to mitigate this challenge and increase confidence in the model.

One approach to mitigate the apparent risk and create more stable result may be to extend the Feature Generation module with additional domain knowledge. As the current system uses historical prices as its sole basis for prediction, it seems natural to extend the system with agents that performs fundamental analysis. In Norway, every company listed on the Oslo Stock Exchange are required by law to issue quarterly financial statements. Financial statements that report higher earnings, increased productivity, etc., are typically accompanied by an increase in the company's stock price. An agent that monitors the stock exchange for financial statements and classifies them as good or bad may thus be very useful. Stock prices are also influenced by expectations caused by news reports. A web-mining agent that tries to analyze if a stock has received positive or negative news coverage may thus provide the model with an important second source of information. Moreover, the global economy and stock markets influences each other in many ways, which might motivate an agent that monitors stock markets in other parts of the world. In order to increase confidence in the generated predictions, an explanation module that provides explanations for the predictions generated would have been a highly regarded addition to the model. Explanation reasoning is a commonly researched topic in the Case-Based Reasoning community (Leake, 1995).

For the risk-averse trader the model may seem too unstable in its present state. However, we still deem the results satisfactory, more so than what was expected when the work was initiated, both in terms of the model architecture and the documented results.

Bibliography

W. Brock, J. Lakonishok, and B. LeBaron. Simple technical trading rules and the stochastic properties of stock returns. *The Journal of Finance*, 47(5):1731–1764, 1992.

S. Browne and W. Whitt. Portfolio choice and the bayesian kelly criterion. *Advances in Applied Probability*, 28(4):1145–1176, 1996.

G. Caginalp and H. Laurent. The predictive power of price patterns. *Applied Mathematical Finance*, 5:181, 1998.

J.-P. Danthine and M. Girardin. Business cycles in switzerland a comparative study. *European Economic Review*, 33(1):31–50, 1989.

K. Downing. Introduction to evolutionary algorithms, 2009. Lecture Notes.

E. Fama. The behavior of stock-market prices. *The Journal of Business*, 38(1):34–105, 1965.

M. E. Fisher. Scaling, university and renormalization group theory. *Springer Lecture Notes in Physics*, Vol. 186, 1983.

D. Floreano and C. Mattiussi. *Bio-Inspired Artificial Intelligence*. MIT Press, 2008.

R. Gencay. Non-linear prediction of security returns with moving average rules. *Journal of Forecasting*, 15(3):165–174, 1996.

R. Gencay. The predictability of security returns with simple technical trading rules. *Journal of Empirical Finance*, 5(4):347–359, 1998.

R. Gencay and T. Stengos. Moving average rules, volume and the predictability of security returns with feedforward networks. *Journal of Forecasting*, 17(56):401–414, 1998.

J. L. Kelly. A new interpretation of information rate. *Information Theory, IRE Transactions on*, 2(3):185–189, 1956.

J. R. Koza, M. A. Keane, and M. J. Streeter. What's AI done for me lately? Genetic programming's human-competitive results. *IEEE Intelligent Systems Magazine*, Vol. 18, 2003.

J. I. Larsen. Automatic trading with artificial intelligence and technical analysis, 2009. Master Thesis Preliminary Project.

D. Leake. Abduction, experience, and goals: A model of everyday abductive explanation. *The Journal of Experimental and Theoretical Artificial Intelligence*, 1995.

W. Leigh, R. Purvis, and J. Ragusa. Forecasting the nyse composite index with technical analysis, pattern recognizer, neural network, and genetic algorithm: a case study in romantic decision support. *Decision support systems*, 32(4):361, 2002.

C. Leser. A simple method of trend construction. *Journal of the Royal Statistical Society. Series B (Methodological)*, 23(1):91–107, 1961.

G. Linløkken and S. Frölich. *Technical stock analysis: for reduced risks and increased returns*. Investtech.com, 2004.

J. D. Lohn, G. S. Hornby, and D. S. Linden. An evolved antenna for deployment on nasa's space technology 5 mission. In U. M. 0'Reilly, T. Yu, R. Riolo, and B. Worzel, editors, *Genetic Programming Theory and Practice II*, volume 8. Springer US, 2005.

T. Mitchell. *Machine Learning*. McGraw Hill, 1997.

J. J. Murphy. *Technical analysis of the financial markets: a comprehensive guide to trading methods and applications*, volume 2. Prentice Hall Press, 1999.

S. Nison. *Japanese Candlestick Charting Techniques: A Contemporary Guide to the Ancient Investment Techniques of the Far East*. New York Institute of Finance, 1991.

C. H. Park and S. H. Irwin. What do we know about the profitability of technical analysis? *Journal of Economic Surveys*, 21(4):786–826, 2007. 206JI Times Cited:5 Cited References Count:168.

L. Rotando and E. Thorp. The kelly criterion and the stock market. *American Mathematical Monthly*, 99(10):922–931, 1992.

H. Shefrin. *Beyond Greed and Fear: Understanding Behavioral Finance and the Psychology of Investing*. Oxford University Press, 2nd edition, 2002.

R. Shiller. From efficient markets theory to behavioral finance. *Journal of Economic Perspectives*, 17(1):83–104, 2003.

A. Shleifer. *Inefficient Markets: An Introduction to Behavioral Finance*. Oxford University Press, 2000.

T. Turner. *A Beginner's Guide to Day Trading Online*. Adams Media, 2nd edition, 2007.

R. E. Walpole, R. H. Myers, S. L. Myers, and K. Ye. *Probability & Statistics for Engineers & Scientists*. Pearson Education International, 8 edition, 2006.

J. W. Wilder. *New concepts in technical trading systems*. Trend Research Greensboro, NC, 1978.

Appendices

A Stocks

Ticker	Name
ACTA	Acta Holding
ACY	Acergy
AIK	Aktiv Kapital
AKER	Aker
AKSO	Aker Solutions
ALGETA	Algeta
ASC	ABG Sundal Collier Holding
ATEA	Atea
AUSS	Austevoll Seafood
BLO	Blom
BWG	BWG Homes
CEQ	Cermaq
COP	Copeinca
DNBNOR	DnB NOR
DNO	DNO International
EDBASA	EDB Business Partner
EKO	Ekornes
FOE	Fred. Olsen Energy
FRO	Frontline
GOGL	Golden Ocean Group
HNB	Hafslund
KOA	Kongsberg Automotive Holding
LSG	Lerøy Seafood Group
MAMUT	Mamut
MHG	Marine Harvest
NAS	Norwegian Air Shuttle
NHY	Norsk Hydro
NPRO	Norwegian Property
NSG	Norske Skogindustrier
OPERA	Opera Software
ORK	Orkla
PGS	Petroleum Geo-Services
PRON	Pronova BioPharma
PRS	Prosafe
RCL	Royal Caribbean Cruises
REC	Renewable Energy Corporation
SALM	SalMar
SAS NOK	SAS AB
SCH	Schibsted
SDRL	Seadrill
SEVAN	Sevan Marine
SNI	Stolt-Nielsen
SONG	Songa Offshore
STB	Storebrand
STL	Statoil
SUB	Subsea 7
TEL	Telenor
TGS	TGS-NOPEC Geophysical Company
TOM	Tomra Systems
VEI	Veidekke
VIZ	Vizrt
WWI	Wilh. Wilhelmsen
YAR	Yara International

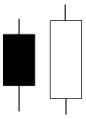
```
import urllib

osebx_tickers = ["ACTA","ACY","AIK","AKER","AKSO","ALGETA","ASC","ATEA","
    AUSS","BLO","BWG","CEQ","COP","DNBNOR","DNO","EDBASA", "EKO","FOE","FRO
    ","GOGL","HNB","KOA","LSG","MAMUT","MHG","NAS","NHY","NPRO","NSG","
    OPERA","ORK","PGS","PRON","PRS", "RCL", "REC", "SALM","SAS-NOK","SCH","
    SDRL", "SEVAN","SNI","SONG","STB","STL", "SUB","TEL","TGS","TOM","VEI",
     "VIZ","WWI","YAR"]

url_start = "http://hopey.netfonds.no/paperhistory.php?paper="
url_end = ".OSE&csv_format=csv"

print "Downloading price history from OSE.."

for ticker in osebx_tickers:
    print "Downloading " + ticker + " ..."
    in_file = urllib.urlopen(url_start + ticker + url_end)
    out_file = open(ticker + ".csv.", "w")
    out_file.write(in_file.read())
    in_file.close()
    out_file.close()
```

Listing A.1: Python script for downloading price data.

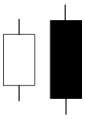
B Candlestick Patterns

(a) Hammer

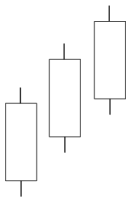
(b) Hanging Man



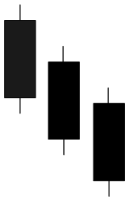
(c) Bullish Engulfing



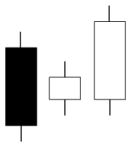
(d) Bearish Engulfing



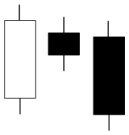
(e) Three White Soldiers



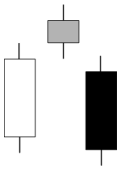
(f) Three Black Crows



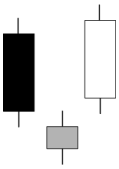
(g) Three Inside Up



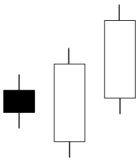
(h) Three Inside Down



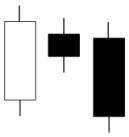
(i) Evening Star



(j) Morning Star



(k) Three Outside Up



(l) Three Inside Down