André Christoffer Andersen
Stian Mikelsen

A Novel Algorithmic Trading Framework Applying Evolution and Machine Learning for Portfolio Optimization

Master's Thesis, Spring 2012

Subject	TIØ4905 - Managerial Economics and Operations Research, Master's Thesis
Supervisor	Professor Alexei A. Gaivoronski
Faculty	Faculty of Social Science and Technology Management
Department	Department of Industrial Economics and Technology Management (IØT)

Abstract

The goal of this thesis is to implement an automated trading system able to outperform the benchmark uniform buy-and-hold strategy. Performance is measured in term of multiple risk and return measures. A comprehensive signal processing framework is built to facilitate rapid development and testing of multiple candidate trading systems. A subset of the most promising trading systems, 14 in total, are presented in this thesis.

The trading systems are not only designed to function as decision support systems for human trader, but also to function as autonomous traders in their own right. They are built to be applicable in real life, any academic value is a welcomed side effect. Many of the trading systems are heavily influenced by machine learning methods, with a special focus on time series prediction. Support vector machines (SVM), artificial neural networks (ANN), and regression are the primary methods applied for low level machine learning. Evolutionary algorithms are applied as high level meta-parameter tuning. In addition to machine learning methods, classical portfolio optimization methods using modern portfolio theory are applied, some in combination with machine learning methods.

The trading systems are tested by means of simulation on data from two stock markets – the OBX index on Oslo stock exchange and Dow Jones Industrial Average (Dow). Results indicate that the Dow index is highly efficient. No trading system was able to outperform the benchmark when using transaction costs. Results on the OBX index indicate some inefficiencies with potential for exploitation. Many trading systems manage to generate higher return than the benchmark. Some achieve excess return with the same level of risk. The efficient portfolio trading system yields highest return given the same level of risk as the buy-and-hold portfolio. In general results tend to support the efficient market hypothesis. Given the same risk level, experiments from this thesis finds that efficient portfolios can outperform uniform ones in inefficient markets as these are not efficient enough in themselves as in the case of OBX. No strategy was found that can substantially outperform an efficient portfolio.

Support Vector Machines are identified to be the best time series predictors among the machine learning techniques used. Evolution is found to be a powerful tool in identifying and training trading systems. Future work will focus on evolution as an integral part of the trading systems.

Sammendrag

Målet med denne oppgaven er å implementere et automatisert system for aksjehandel. Systemet skal være i stand til å utkonkurrere referanseporteføljen som bruker en uniform kjøp-og-hold strategi. Ytelse måles i form av risiko og avkastning. Et omfattende signalprosesseringsrammeverk er laget for rask utvikling og testing av ulike aksjehandlesystem. En undergruppe av de mest lovende aksjehandelssystemene, 14 i alt, er presentert i denne avhandlingen.

Systemene er ikke bare designet for å fungere som beslutningsstøtte-system, men også for å fungere selvstendig opp mot en børs. De er bygget for praktisk bruk, med akademisk verdi som en nyttig bivirkning. Flere av systemene er sterkt påvirket av maskinlæringsmetoder, med spesiell fokus på tidsserierprediksjon. Støttevektormaskiner (SVM), kunstige nevrale nettverk (ANN), og regresjon er de primære metodene som brukes for maskinlæring på lavt nivå. Evolusjonære algoritmer brukes som meta-parameter-læring på høyere nivå. I tillegg til maskinlæringsmetoder er klassiske porteføljeoptimaliseringsmetoder fra moderne porteføljeteori anvendt. Noen av systemene bruker begge methodene.

Systemene er testet ved hjelp av simuleringer på data fra to aksjemarkeder – OBX-indeksen på Oslo børs og Dow Jones Industrial Average (Dow). Resultatene indikerer at Dow-indeksen er svært effisient. Ingen system var i stand til å utkonkurrere referanseporteføljen når transaksjonskostnader var medregnet. Resultater på OBX-indeksen indikerer noe ineffisiens med potensial for utnyttelse. Mange handelssystemer klarer å generere høyere avkastning enn referanseindeksen. Bare noen oppnår meravkastning med samme grad av risiko. Systemet som bruker metoder for konstruksjon av effisiente porteføljer gir høyest avkastning, gitt samme risiko, som kjøpe-og-hold porteføljen. Generelt viser resultatene en tendens til å støtte hypotesen om effisiente markeder. Gitt samme risikonivå vil kun effisient porteføljer utkonkurrere kjøpe-og-hold strategien, og da kun i tilfeller hvor den i seg selv ikke er effisient som i OBX. Ingen strategi er funnet som kan utkonkurrere en effisient portefølje. Støttevektormaskiner er identifisert til å være den beste tidsserieprediktoren blant maskinlæringsmetodene som brukes. Evolusjon markerer seg som et kraftig verktøy for å identifisere og stille inn predikasjonsystemer. Fremtidig arbeid vil fokusere på evolusjon som en integrert del av aksjehandelsystemene.

Preface

This document and its associated source-code is the end product of the Masters Thesis of André Christoffer Andersen and Stian Mikelsen at the Norwegian University of Science and Technology (NTNU). The thesis work lasted from January to June 2012.

André Christoffer Andersen is a Master's student of the Department of Industrial Economics and Technology Management (IØT) specializing in Applied Economics and Optimization. Stian Mikelsen is a Master's student of the Departments of Computer and Information Science (IDI) specializing in Intelligent Systems.

Exploring the fringes of the interdisciplinary field of algorithmic portfolio optimization has been both challenging and enlightening. If applying knowledge and methods from investment theory, optimization and machine learning has taught us anything, it is that it requires a supporting and dedicated thesis committee. The authors would like to thank *their* thesis committee for the invaluable feedback and open doors they have offered in this ambitious endeavor.

The thesis committee consists of Alexei Gaivoronski, professor of finance and optimization at IØT and Helge Langseth, professor of intelligent systems at IDI.

Thank you.

Trondheim, December 12, 2012

André Christoffer Andersen	Stian Mikelsen
Master's Student at IØT	Master's Student at IDI

Contents

List of Figures

List of Tables

Chapter 1

Introduction and Overview

1.1 Introduction

Algorithmic portfolio optimization is an interdisciplinary endeavor intersecting many sciences. This field is approached from the domains of Investment Theory, Optimization and Machine Learning.

Machine Learning is the science of autonomous decision making based on empirical data. **Mathematical optimization** is the mathematical approach for finding optimal solutions given a set of alternatives. Applying these to the problem domain of **investment theory**, the science of investment allocation, a comprehensive algorithmic portfolio optimization framework is designed.

The aim of this thesis is to make an autonomous trading system based on historical data without neglecting the fundamental principles of modern portfolio theory. In order to identify a good approach, 14 different approaches are implemented and evaluated.

This paper argues that the market tends to be efficient. However, that given the right application of modern predictive tools it is possible to uncover underlying patterns that are too obscure for the general market to take into account.

Using heuristics, e.g. evolution and machine learning, in conjunction with practical classifiers and time series prediction methods, this paper aims to build a robust and adaptive system that is able to outperform the financial market it operates on. The project is engineer-oriented rather than scientifically descriptive, though scientific principles are applied for system- and performance evaluation.

1.2 Background and Motivation

1.2.1 The Problem Domain

This thesis' problem domain concerns investment allocation, with the aim to build an autonomous trading system that can exploit possible market inefficiencies in order to generate higher return at the same, or lower, risk than the market it operates in.

Speculative trading in an effort to "beat the market" is as old as open markets themselves. Active investment management is at the time of writing a huge business involving trillions of dollars world-wide. While being a well-established industry it is not without its share of misconceptions and controversy. The main motivation behind active investing is the prospect of generating as high of a return as possible given an acceptable level of risk. Traditionally, human traders tend to use subjective approaches for portfolio selection where there are no universally agreed upon best practice.

The advent of quantitative trading and later algorithmic trading has done little to settle the controversy. The appeal of automated trading systems that can replace emotionally constrained human financial analysts has motivated many a research project. There are numerous reasons for wanting to automate trading. One can achieve more trades, faster and with less human biases. The brunt of more advanced commercial systems focus on **high frequency trading**, using short portfolio holding periods. Some of these focus on acquiring market data first in order to reap the benefit of information superiority, a practice called **low-latency trading**.

In general there are now many machine learning methods that have shown reliable results on stochastic problem domains. Especially promising are hybrids that combine multiple methods. However, as with classical probability theory[148] in ancient times[1], trading methods today are highly proprietary. The proprietary nature of the financial problem domain makes it hard to find successful benchmark trading algorithms for evaluation. If anyone actually achieves profitable results there is no incentive to share them, as it would negate their advantage.

The whole foundation of active investing is criticized by some theorists claiming that markets are efficient and thus act like mere "random walks"[58; 59]. However, others point to underlying psychological patterns which could justify an active investment strategy [25]. For any system to actually work there must be some

[1]It is speculated that breakthroughs in the field of probability theory has happened several times, but never shared. This is due to its practical application in gambling.

degree of market inefficiency. This is discussed in depth in 2.1 Financial and Investment Theory.

1.2.2 The Methodology

There are two main paradigms of asset analysis, fundamental and technical. In short, **fundamental analysis** looks at qualitative features of an asset in order to assess its intrinsic value. **Technical analysis** relies on price data only and is thus a pure quantitative measure based on historical price data. Traditionally, most trading algorithms have focused on the latter. Both methods are widely used in practice, often in conjugation with each other. Technical analysis is for the main part used in this thesis due to its applicability for automated trading. Still, a more fundamental analysis is an important aspect of for future work.

This thesis attempts to use concepts from machine learning in order to create an information ecosystem that is capable of working with a disparate set of information sources. Rather than being fast, this system is intended to focus on extracting latent information in the market that is non-obvious, yet valuable. To the authors' knowledge, this is an under-explored feature in applied systems.

The basic units of the mentioned ecosystem are interconnected processor nodes. Data from relevant data sources are passed through a network of these nodes ending up at some sink, may it be the computer screen of a trader or a stock market network interface. The nodes are fed several streams of temporal data, known as time series. These are used to output a portfolio distribution for any given point in time in the series. The performance is evaluated on a simulated market. This makes it possible to uncover what strategies have better promise.

All nodes are not created equal in these networks. Some nodes have the simple task of calculating rate of change of asset prices while others might be tasked to construct entire portfolios based on a moving segment of historical data. These more advanced nodes are similar in nature to the concept of intelligent agents, defined as "a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." [151, pg. 15]. What sets this approach apart from earlier work is the ability to combine many different commonly used strategies into one collectively larger and more robust system. A collection of nodes tasked with creating portfolios can be viewed as a bounded entity called a trader algorithm, system or module. The design of various traders will reflect an effort to maximize return while controlling risk. Some nodes of a trader will focus on making time series forecasting, others will make sure portfolios are valid and enforce fail safe procedures. These are non-trivial problems that require many technical and financial

questions to be resolved.

Special focus will be given to biologically inspired methods due to their good pattern recognition capabilities for stochastic problems with underlying latent information[23]; a key characteristic of temporal financial data. One class of these methods are artificial neural networks, which show many promising results in the financial domain[87]. Alternative methods such as support vector machines[126], auto regression and simple technical analysis methods will also be applied.

Evolutionary algorithms are used to set many of the meta-parameters[2] that define the nodes, while some of them will be set manually when suitable. This evolutionary focus is motivated by promising result for evolving neural network architecture[153]. Among the portfolio producing nodes the best, or a combination of several top performers, will be chosen. Another component in the system then decides whether it is worth it to rebalance the portfolio.

The whole process will proceed in the following steps:

1. Train and evolve nodes based on market data
2. Form a portfolio based on input from nodes
3. Decide whether to rebalance the portfolio position

1.2.3 Personal Motivation

Intelligent systems that perform tasks autonomously is a fascinating field of computer science. The application of this field's methods to the problems deemed hardest for procedural computation is both rewarding on a personal level, and important to many fields concerned with decision support. Few problem domains are as hard for intelligent systems as making money on financial markets. A hard problem is often measured by the degree of imperfect information and stochasticity (see figure 1.1). Modern portfolio theory, with its efficient market hypothesis (EMH), states that financial markets have perfect information revealed in the market price[101]. This reduces the problem to a find-the-efficient-frontier problem. Critics, such as financial behaviorists, say that the problem is more complex as it has elements of imperfect information and psychology[91]. This is evident in market anomalies of financial crises. Given the latter view, the problem is much more applicable to advanced heuristic and machine learning methods. The fact that this is not a settled question makes the subject interesting, and motivates the authors' effort into the problem domain.

[2]Meta-parameters: A parameter that defines an object containing (sub)parameters

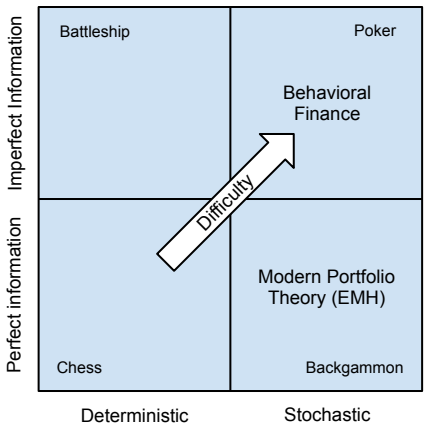


Figure 1.1: The difficulty for a computerized solution to a problem can be measured in its level of imperfect information and stochasticity. High level of both constitutes difficulty, but also opportunity.

1.3 Goals

This section will present the goals and sub-goals of the thesis.

> *The **main goal** of this thesis is to design a novel algorithmic trading system that creates optimized portfolios, exploiting market inefficiencies in order to generate a statistically significant better return than the simple market average, at equal or lower risk.*

The main goal has been divided into three main categories of subgoals

1. System Architecture Requirements
2. Technological Feasibility
3. Financial Feasibility and Requirements

1.3.1 System Architecture Requirements

The system must handle seamless module exchange

The system should handle exchange of different combinations of technologies and methods. In order to do this seamlessly the system needs a modular architecture that allows change of approach post-compilation. This motivates the use of architecture with an underlying general framework, rather than a single hard-coded implementation. This should increase both the testability and extensibility of the system. Modularity and testability is the main attributes used to evaluate the

architecture with. The system will consists of several interconnected components, each using different state of the art technologies. The ability to extensively test each of these components separately will be crucial to prevent complex run-time bugs.

The system should be scalable and parallelizable

Due to the inherent complexity of the system computational performance and tractability can become an issue. This can be improved be writing efficient code, as well as look at strategies for scalability and parallelization.

The system must be possible to implement in less than four months

The actual system implementation will be done as a part of a half-year Master's degree thesis. The architecture size and complexity must therefore take this time limit into account. This requires scheduling and setting natural milestones to evaluate progress.

1.3.2 Technological Feasibility

Identify time series analysis methods

The aim is to analyze time series with technical methods. A central part of the project is to identify which performs best. The focus is on machine learning and evolution, but also here are there many variants. To decide on which combination is best is an important part of this project.

Compose a system training scheme

The system should be robust and adaptable. This is achieved by employing learning, i.e., parameter optimization, on the system. Some of the parts like the ANNs are not feasible without learning. An important goal is to discover what needs learning and what methods to use.

Ensure robustness and adaptivity

In order to achieve certain credibility as a stable system it is important to have a certain degree of both robustness and adaptivity. In this paper robustness is used to describe the ability to handle abnormal input, while adaptivity describes the ability to adapt to new environments. Robustness in this case is in regard to the market data and adaptivity to the sometimes rapidly changing financial markets.

1.3.3 Financial Feasibility and Requirements

Identify appropriate input data for the system

Deciding which input data to use is of paramount importance. No matter how well a system is designed, or how good the underlying theory is, if you feed it garbage as input you get garbage as output. This is why the system needs to identify high quality information sources based on financial criteria.

Return and risk should be incorporated in the resulting portfolio advice

Though the natural aim of an algorithmic trading system is to maximize return, it is not done without perils. Securities with a high return have a tendency to entail high risk. This risk-return trade-off is an ingrained feature of modern investing. The system must therefore be able to incorporate a degree of risk aversion as well as risk reducing behavior. The aim is to discover a hidden, more correct, risk-return relationship which the system can optimized.

Identify how to evaluate portfolios for training the system

During training the various learning modules need some sort of benchmark portfolio to base its performance evaluation on. This portfolio is supplied by some always-correct supervisor that knows how the system ought to have acted at any given time. It is important to understand that there is no trivial way to identify such an optimal supervisor portfolio. The supervisor in itself can be taught of as a high-level parameter that should reflect the trader's risk-return preferences.

1.4 Contributions

This thesis creates a framework for building and testing automates trading systems. 14 different trading systems are then built and tested in a search to find a system that can outperform a uniform buy-and-hold strategy. To reduce uncertainty the experiments are run on both the Dow Jones Index and on the OBX index at Oslo Stock Exchange. No trading system is identified to be superior to the benchmark on the Dow dataset, indicating that this truly is an efficient market. On OBX the combination of an efficient frontier with an SVM is found to perform best. In addition several technical analysis approaches are found to give excess returns. These results indicates that the OBX and thus Oslo Stock Exchange is less efficient then Dow and that anomalies tend to persist long enough to be capitalized.

1.5 Research Method

The main question in this thesis is whether a trading system can be built that consistently beat the market by employing machine learning techniques. The trading systems are built as modular structures of techniques that in the end produce a portfolio. They are meant to be able to run online, integrated to an actual stock market, though this integration is not part of the scope of this thesis. The point is not to test whether one specific design is feasible, more broadly, the whole concept of data mining techniques for portfolio selection. A framework is built to facilitate the construction of such systems. A special focus is put on the use of machine learning and evolution.

A set of trading systems is defined. These constitute a complete network of nodes which together form a complete unit capable of autonomous trading. A simple example of this is the Efficient Frontier trading system which finds a mean-variance optimized portfolio given a return target.

The systems are evaluated on a simulated market based on historical data. The trading systems are evaluated against two different indices, The Dow Jones Industrial Average, and OBX, which consists of the 25 most liquid companies at Oslo Stock Exchange. The average daily mean, variance and cumulative return the trading systems achieve are evaluated against a benchmark portfolio. This benchmark is a uniform portfolio of all the available stocks. This portfolio is never rebalanced, and as such uses a buy-and-hold strategy. It is not a goal to come close to high frequency trading, and daily data is the highest resolution used.

1.6 Thesis Structure

This thesis is a continuation of a project from the fall of 2011 [14]. Much of the background search and design work for the system presented in this paper were completed in that project. To present the whole system in a structured manner many of the results from that project are reused in this thesis. This thesis is written as if the two projects were one whole. The Introduction and Overview as well as Theory and Background are largely improved versions from the former paper while the rest is new.

The next chapter introduces the theory and background required to understand the rest of the thesis. Chapter 3 presents the framework created in this project. The actual trading systems created are introduced in chapter 4. Chapter 5 describes the experiments and presents their results along with some analysis. In chapter 6 the whole thesis is summed up along with some more in-debt discussion. Chapter

6 concludes this thesis. Some additional information is gathered in the appendix.

Chapter 2

Theory and Background

2.1 Financial and Investment Theory

2.1.1 Financial Markets

Financial markets are mechanisms that allow people to trade liquid and fungible assets at relatively low transaction costs. Assets are liquid in the sense that they are "easily converted into cash."[1], and fungible in the sense that they can be "replaceable by another identical [asset]" and are "mutually interchangeable." [1]. Financial markets facilitate raising capital, transfer of risk, transfer of liquidity and international trade. Capital is raised by acquiring investors. Risk can be spread, or transferred, by investing in uncorrelated assets. Insurance is a common example of transfer of risk between parties. There are several kinds of markets, of which some more important are shown in table 2.1.

Stocks. Stocks are instruments that represent an ownership share in the equity of a company. One can say a stock is divided into multiple shares, but for semantic purposes these words are equivalent [1]. The stock market is a public entity for the trading of such company stocks. Stocks are frequently traded on the stock market both for investment purposes and speculation.

When a company earns a profit, the owners can either let it remain in the company as a reinvestment, or choose to withdraw all or a portion of it. Companies basically have two ways to pay back profits to the stock holders. They can buy back shares or pay dividend. When a company pays dividend, each stock holder receives a fixed amount per share, and the equity of the company is reduced according to the amount paid out. As stocks are connected to companies there is always the risk of them losing their value in the event of a bankruptcy. In a company bankruptcy the

Table 2.1: This is a lists the different financial markets and briefly explains their purpose

Market	Description
Bond Markets	Provides financing through bonds
Stock Markets	Provides financing through stocks
Commodity markets	Facilitates trading of commodities
Money Markets	Provides short term debt financing and investment
Insurance markets	Facilitates redistribution of various risks
Futures Market	Provide standardized forward contracts for trading products
Foreign Exchange Markets	Facilitates trading of foreign exchange (currency)
Derivatives Markets	Provide instruments for the management of financial risk

bank and other lenders have first claim to all remaining value, while stock holders are first paid after all debt has been settled. This makes stocks inherently more risky than to lend out money, and is also one of the main reasons one can expect more return from such investments.

Stocks listed on the market can change value drastically for many reasons. In essence the company is valued based on expected future yields. The fluctuations arise from events affecting the company or market, some theories that it even has a reflective property where price changes affect price levels in a feedback loop. These are properties that are expected in day-to-day trading. However, there are some price changes that come from adjustments that are exceptions to the rule, and will create trouble for an automated trader. Two are already mentioned, dividend pay-outs and bankruptcy. Other issues may be splitting of companies, mergers or de-listing. These and dividend are no problem for "human" stockholders as they will receive the dividend money and while their stock price drops accordingly. However an automated trader needs some way to understand that dividend is not a loss, even though the price drops significantly, i.e., a dividend payout should not make an automated trader panic.

Bonds. Bonds are traditionally considered to be a low return-risk option. These are often used as a supplement in portfolio building and as a safe haven in troubled

times. Recent times have seen quite high bond yields from certain countries, but this is reflected in the sudden increase of risk these bonds are predicted to have. Still, in normal circumstances bond tend to have low returns, and their main benefit would be as a safe haven.

Commodity. Commodity markets differ vastly depending on which commodity is in question. Each has their own set of influential parameters that affect their price. Some of these, like gold, are used as safe havens when other more risky markets are volatile. Commodities like oil or copper are sometimes seen as a predictor of future prospects. An increase in the price of copper have even been claimed to signal future rise of the stock market, though this seem to be an extinct phenomenon[4].

Forex. Forex or foreign exchange trading is trade between different currencies. This is an inherently speculative zero-sum game as all changes are relative to the other currencies. It is a popular market for speculation with many large professional actors.

Derivatives. Derivatives are contracts between two parties that derive their value from some underlying asset. Some common derivatives are options, futures or forwards. The reason for such derivatives is for actors to be able to decide the exact ratio of risk and return and on which asset they want to "bet". Derivatives are typically used to either speculate through certain "bets" or to hedge risk (insurance).

2.1.2 Trading

Most assets in the world are traded. Where there is trade, there can be speculation. In finance, speculation is a financial action that does not promise safety of the initial investment along with the return on the principal sum [30]. This is contrary to an investment. Speculation is often called active trading and is usually done with a short time horizon. The type of trading this system does is to be considered speculation as the plan is to generate returns by buying and selling positions in assets relatively often.

The Virtue of Speculation

One concern that might be raised about active trading is of social nature. Some claim that pure speculation is just a form of institutionalized parasitism that brings no value to society. However, this does not take into consideration a wider perspective. Businesses need risk capital to grow. Few investors are willing to lock their capital down for as many years as the company would need it. With a liquid financial market this can be mitigated by giving investors the ability to get rid of

their exposure in the company at any given time. Speculation increases the liquidity of financial markets, thus attracting investors to growth businesses that need time to mature. In turn, this ushers forward bottom up growth and puts pressure on existing companies to stay profitable and innovative. Furthermore this increase in trading activity will in itself make the markets more efficient as more or less rational investors will correct all arbitrage options they might find.

Categories of Traders

There are many different types of traders active in the various markets. They are often categorized by their trading time-frame. Alternatively it is possible to categorize based on what kind of strategies they apply. The most common will be introduced and presented with the field of algorithmic trading as a backdrop.

In general there are some main distinctions to make. There is difference between technical and fundamental trading. Fundamental trading includes factors outside the price of the security; typically a company's accounts are used. Technical trading however is only based on the actual price and its qualities. Technical traders base their strategy on the assumption that historical price data can in some way predict future prices. These aspects will be further elaborated on in 2.2. The degrees of which analysis one employs affect the time-frame, as fundamental analysis are inherently much more static than technical. One can get a price update several times per second, while the company results are only published once a quarter. Fundamental data are also more qualitative and thus harder to analyze quickly, e.g. news. Analysis of technical data is purely quantitative, and can thus be handled by computers. Generally the shorter time-frame, the more automatic are the traders. Additionally the risk is can be seen higher as the market converges toward a zero-sum game when the time-frame considered decreases[1].

Position Trading. The longest time-frame is often called position trading. It generally consists of all positions held longer than a few months. This is a very reputable strategy, recommended by Warren Buffet himself[128], but with a limited possibility of return. Since one has plenty of time to do a thorough analysis of the asset, algorithmic methods have less advantage here, but can be used as supplementary guidance. This category also includes more active investors, who are active in a different sense than earlier described. They take an active part in management of the company they're invested in, like private equity companies.

Swing Trading. Positions held over days or weeks include strategies like swing trading and more short-term fundamental analysis strategies. This is generally

[1]The long term increase called the equity premium[106] does only apply if one holds a position for some time. Trades where one aims to hold a position as short as possible will thus in principle be a zero-sum game as one trades against a snapshot in time where all values are fixed.

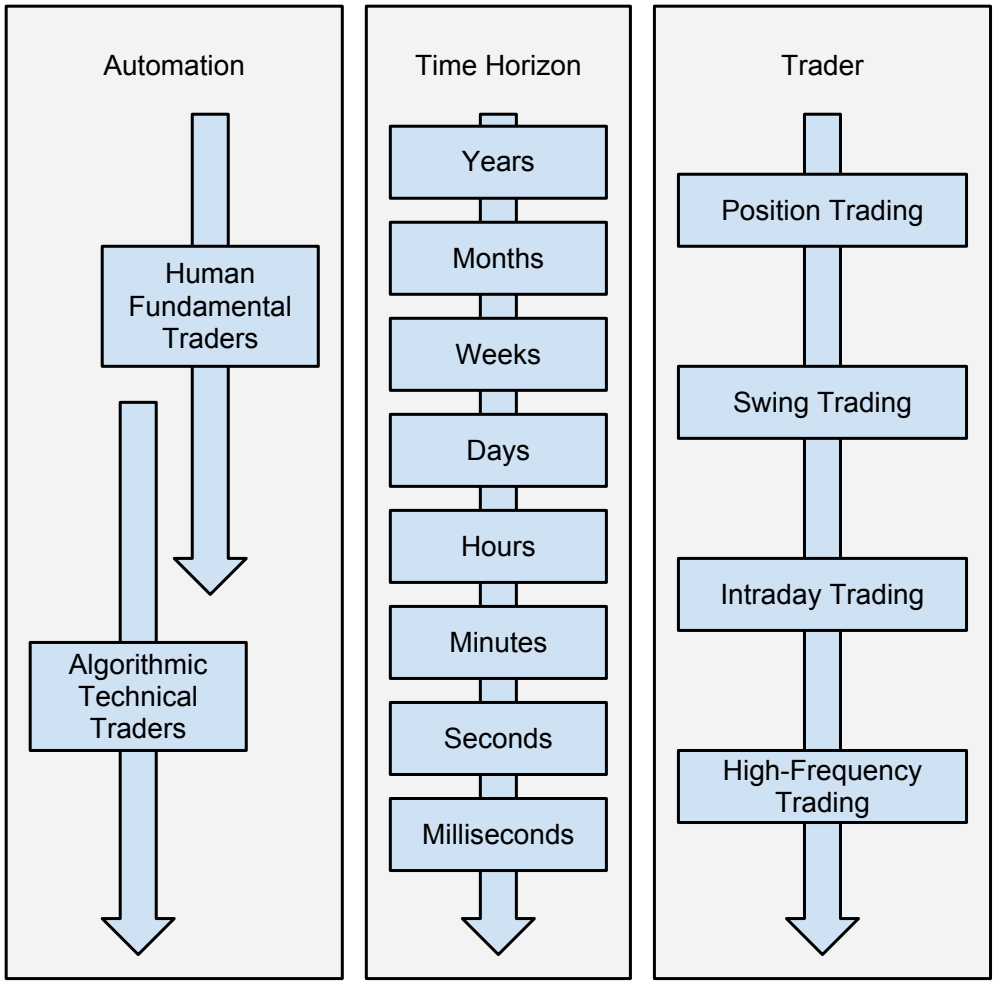


Figure 2.1: Trading time horizon and automation.

the shortest time-frame one finds purely fundamental analysis traders. Changes in price on a lower scale than days are often considered noise to these traders. There is some technical analysis done over several days, but it is limited. This is because the primary goal of pure technical analysts is to speculate on price swings. A shorter time frame will give more than enough volatility to do this, due to the fractal[2] nature of the stock market.

Intraday Trading. Intraday trading is trading confined to a single day, most day-traders sell their positions before close. This is mainly speculation and mostly done by technical analysis, though one might supplement with fundamental techniques.

[2]A fractal is a mathematical concept where roughness exist at whatever resolution one uses. The graph of price a different time periods will tend to have roughly the same kind of volatility.

Day trading can be a job for human investors, but require much time and is quite close to a zero sum game due to the lack of the long term equity premium. Intraday trading is becoming the realm of machines as they can search through much more data. Most machines are however trading on more or less instant strategies where the aim is to hold an actual position as short as possible; these are called High-frequency traders.

High-Frequency Trading. High-frequency trading (HFT) is a poorly defined term. In this paper it is used in its semantic meaning of all trading done at a high frequency instead of meddling with all the contradictory definitions in the literature. With this meaning of the term, HFT will be a broad concept that includes all trading where the trading speed is used as a competitive advantage. In 2010 these systems trade on milli- and microseconds[75]. All HFT is done by algorithms, as the speeds required for this trading type are beyond human capabilities.

Low-Latency Trading. Low-latency trading is a subclass of HFT, and can be thought of as a specific strategy of HFT. These are a naive set of simple algorithms which sole advantage is their ultra-low latency, direct market access. An example of a low-latency strategy is to survey the news. The instance good news hits the market, a low-latency trader will buy from slower sellers which have not had time to increase their bid price, and then sell to a new higher price. Most of these tactics revolve around quickly finding and exploiting arbitrage possibilities. There is tough competition in having the fastest algorithms and the lowest latency. This activity is very capital intensive due to the hardware requirements and it needs a certain initial capital. There are few amateurs in this area and most actors are large banks, funds or even specialized algorithmic trading firms[32].

2.1.3 Modern Portfolio Theory

Modern portfolio theory (MPT), introduced by Harry Markowitz in 1952 [101], is a theory which attempts to create portfolios of assets by striking a balance between return and risk. In the basic terms, one can maximize the portfolio expected return given a certain risk, or equivalently minimize risk for a given level of return.

One of the main ideas of MPT is that a portfolio of assets should be composed not based on the performance of individual assets, but rather taking a holistically view of asset performance. This means that the internal risk-return dynamics of portfolios must be taken into consideration when evaluating them. The classic mathematical model of MPT uses **mean return** as a measure of **expected return** and the **return variance** as the **risk** measure.

Mean Return and Variance

Mean return and variance are regarded as the two primary return and risk measures of any asset or collection of assets. These measures were used by Markowitz when he originally presented modern portfolio theory. To calculate them we start with asset prices. Given $T \in \mathbb{N}_1$ time periods and $n \in \mathbb{N}_1$ assets we can define the price time series of all assets in question as a matrix $\mathbf{P} \in \mathbb{R}_{\geq 0}^{T+1 \times n}$ consisting of price entries $p_{ti} \in \mathbb{R}_{\geq 0}$, one for each asset $i \in \{0, \cdots, n\}$ at time period $t \in \{0, \cdots, T+1\}$. The row vectors of $\mathbf{P}$ can be viewed as price snapshots in time, or datapoints in a dataset, concepts that will be covered in section 2.3.2. From the price entries we can define the temporal (rate of) return development matrix $\mathbf{R} \in \mathbb{R}^{T \times n}$ with entries $r_{ti} \in \mathbb{R}$ by equation (2.1).

$$r_{ti} = \frac{p_{ti} - p_{(t-1)i}}{p_{(t-1)i}} \tag{2.1}$$

Mean Return. The r_{ti} entries can be collected into n return column vectors $\boldsymbol{r}_i \in \mathbb{R}^T$ each representing a return time series of an asset i. This makes it possible to define the mean return for a single asset as in equation (2.2), which can be collecting into a mean return vector $\bar{\boldsymbol{r}} = [\bar{r}_1 \ \cdots \ \bar{r}_n]^{\mathrm{T}} \in \mathbb{R}^n$ for all assets. A more succinct way to define the mean return vector is by matrix notation as in equation 2.3.

$$\bar{r}_i = \frac{1}{T} \boldsymbol{r}_i^{\mathrm{T}} \mathbf{1} \tag{2.2}$$

$$\bar{\boldsymbol{r}} = \frac{1}{T} R^{\mathrm{T}} \mathbf{1} \tag{2.3}$$

Return Variance. For any two time series return vectors $\boldsymbol{r}_i$ and $\boldsymbol{r}_j$, and their means $\bar{r}_i$ and $\bar{r}_j$ we can compute their covariance using (2.4).

$$\operatorname{cov}(\boldsymbol{r}_i, \boldsymbol{r}_j) = \frac{1}{T} (\boldsymbol{r}_i - \bar{r}_i)^{\mathrm{T}} (\boldsymbol{r}_j - \bar{r}_j) \tag{2.4}$$

Using this it is possible to retain all possible covariance computations in a symmetric covariance matrix $\boldsymbol{\Sigma}$ as defined by (2.5). Alternatively, it is possible to

show that the covariance matrix can be computed directly by (2.6).

$$\boldsymbol{\Sigma} = \begin{bmatrix} \operatorname{cov}(\boldsymbol{r}_1, \boldsymbol{r}_1) & \cdots & \operatorname{cov}(\boldsymbol{r}_1, \boldsymbol{r}_n) \\ \vdots & \ddots & \vdots \\ \operatorname{cov}(\boldsymbol{r}_n, \boldsymbol{r}_1) & \cdots & \operatorname{cov}(\boldsymbol{r}_n, \boldsymbol{r}_n) \end{bmatrix} \tag{2.5}$$

$$\boldsymbol{\Sigma} = \frac{1}{T}(R - \frac{1}{T}\mathbf{1}\mathbf{1}^{\mathrm{T}}R)^{\mathrm{T}}(R - \frac{1}{T}\mathbf{1}\mathbf{1}^{\mathrm{T}}R) \tag{2.6}$$

Expected Return and Risk of a Portfolio. Before we can state the various versions of the MPT optimization problem we need to review how to measure the portfolio expected return and risk as a function of mean return and variance. Let $\boldsymbol{w} \in [0, 1]^n$ with the constraint $\mathbf{1}^{\mathrm{T}}\boldsymbol{w} = 1$ be the distribution over n assets determining how much should be invested in each asset, then the return r_p from such a portfolio distribution can be computed by (2.7).

$$r_{\boldsymbol{w}} = \bar{\boldsymbol{r}}^{\mathrm{T}}\boldsymbol{w} \tag{2.7}$$

The variance of the portfolio can be computed from a doubly weighted covariance matrix as shown in (2.8).

$$\sigma^2_{\boldsymbol{w}} = \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{w} \tag{2.8}$$

Expressing MPT as a Mathematical Model. There are different ways to express MPT as a mathematical optimization model, each with subtle differences. What we want is to maximize $r_{\boldsymbol{w}}$ or minimize $\sigma^2_{\boldsymbol{w}}$ by adjusting the portfolio distribution $\boldsymbol{w}$. However, since these are two concurrent objectives we cannot optimize them at the same time. What we can do is fix one at a desired level and optimize the other. The optimization task of 2.9 illustrates constraining risk and maximizing return, while 2.10 illustrates constraining return and minimizing risk. Notice the lower bound constraints on the portfolio $\boldsymbol{w} \geq 0$. Removing these will allow the optimization task to short stocks; however, this behavior is outside the scope of this thesis. The two optimization problems can be solved using Lagrange multipliers thus reducing the problem to a quadratic, i.e., convex, optimization

problem.

$$\begin{array}{rlcl} \max_{\boldsymbol{w}} & r_{\boldsymbol{w}} & = & \bar{\boldsymbol{r}}^{\mathrm{T}}\boldsymbol{w} \\ \text{s.t.} & \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{w} & \leq & \sigma_p^2 \\ & \mathbf{1}^{\mathrm{T}}\boldsymbol{w} & = & 1 \\ & \boldsymbol{w} & \geq & 0 \end{array} \tag{2.9}$$

$$\begin{array}{rlcl} \min_{\boldsymbol{w}} & \sigma_{\boldsymbol{w}}^2 & = & \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{w} \\ \text{s.t.} & \bar{\boldsymbol{r}}^{\mathrm{T}}\boldsymbol{w} & \geq & r_p \\ & \mathbf{1}^{\mathrm{T}}\boldsymbol{w} & = & 1 \\ & \boldsymbol{w} & \geq & 0 \end{array} \tag{2.10}$$

Applying a Model of Risk Aversion. An alternative to fixing one of the performance measures is to apply an utility function $U(\boldsymbol{w})$ that converts the two dimensional performance measures to a single dimension. There are many ways to do this, however, a main constraint is that it must conform to **rationality**[3]. Figure 2.2 illustrates how different portfolios in the two dimensional risk-return space can be mapped to a one dimensional utility. Notice that some internal requirements to this mapping. From the perspective of portfolio A, we can qualitatively see that it stochastically dominates portfolio C, but without a utility function we cannot say anything about A's ordering compared to B and D. Likewise we see that portfolio A is stochastically dominated by portfolio E.

The utility function of (2.11) is widely used linear objective function for the MPT problem. It incorporates a risk aversion factor ρ that weights the risk component of a linear combination of the two performance measures.

$$U(\boldsymbol{w};\rho) = U\left(r_{\boldsymbol{w}}, \sigma_{\boldsymbol{w}}^2;\rho\right) = \bar{\boldsymbol{r}}^{\mathrm{T}}\boldsymbol{w} - \rho\boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{w} \tag{2.11}$$

The resulting optimization problem is also quadratic and solvable in reasonable time. Using the utility function of (2.11) as an objective function we get the

[3]A simple definition of rationality says that rational agents wanting more rather than less of a good. Of course this is a gross simplification of rational choice theory, yet it will suffice due to being outside of the scope of this thesis.

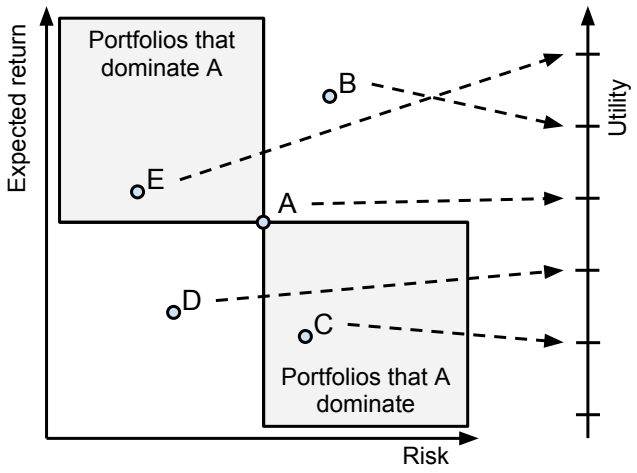


Figure 2.2: With ordinal utility we can say that some portfolios stochastically dominate others, however, for some portfolio pairs only a utility function can settle the preference. For A in the example above the ordinal utility says that $E > A > C$, but nothing about B and D relative to A. A utility function will remove any ambiguity, and in this case states that $E > B > A > D > C$ which is a special case of the ordinal utility.

optimziation problem of (2.12).

$$\begin{array}{lrcl} \max_{\boldsymbol{w}} & U(\boldsymbol{w}) & = & \bar{\boldsymbol{r}}^{\mathrm{T}}\boldsymbol{w} - \rho \boldsymbol{w}^{\mathrm{T}}\boldsymbol{\Sigma}\boldsymbol{w} \\ & \mathbf{1}^{\mathrm{T}}\boldsymbol{w} & = & 1 \\ & \boldsymbol{w} & \geq & 0 \end{array} \tag{2.12}$$

The Efficient Frontier. In the optimization problems above the user must choose some parameter σ_p^2, r_p or ρ which reflects their preference. A way of analyzing the effect of changing the parameters is to look at the **efficient frontier** characteristics of the problems. The efficient frontier was coined by Harry Markowitz[101], and can be defined by the optimal solution space, i.e., optimal portfolios $\boldsymbol{w}$, that exist for all parameter settings that yield feasible solutions. It can be visualized as an upper-half hyperbola in the mean-variance space created by the optimal portfolios, see figure 2.3. Efficient portfolios are portfolios with the highest risk-return ratios given any parameter setting. This means that it is always possible to select a portfolio in the efficient frontier that stochastically dominates any non-efficient portfolio.

Adding a Risk Free Asset. Introducing a **risk free asset** creates a special case where the efficient frontier becomes linear between the risk free asset and

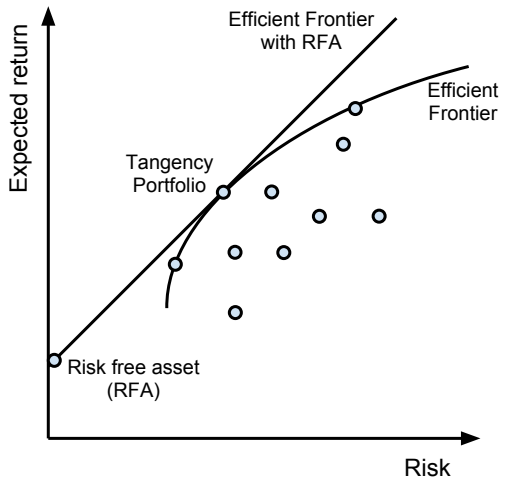


Figure 2.3: Two efficient frontiers, the hyperbolic frontier does not have a risk free asset, while the linear does.

the **tangency portfolio**. The tangency portfolio is a special portfolio which if used with the risk free asset can produce risk-return trade-offs which stochastically dominate any other efficient portfolio lacking the risk free asset. Portfolios that have mean return less than the tangency portfolio lend money at the risk free rate, while portfolios that have larger mean return than the tangency portfolio borrows at the risk free rate. In this thesis borrowing and shorting is not allowed.

Hedging
Diversification

Efficient Market Hypothesis

The **efficient market hypothesis** (EMH) was first developed by Fama in his PhD thesis during the 1960s [57; 59]. The hypothesis asserts that financial markets are informationally efficient, in other words, prices reflect all relevant information. This means that changes in price are reactions to unpredictable events, i.e., price movements are nothing more than random walks. The EHM assumes that all actors in the market act rationally according to Expected Utility Hypothesis [145].

There are three major versions of EMH:

1. Weak EMH claims that prices on traded assets already reflect all past publicly available information.
2. Semi-strong EMH claims both that prices reflect all publicly available information and that prices instantly change to reflect new public information.

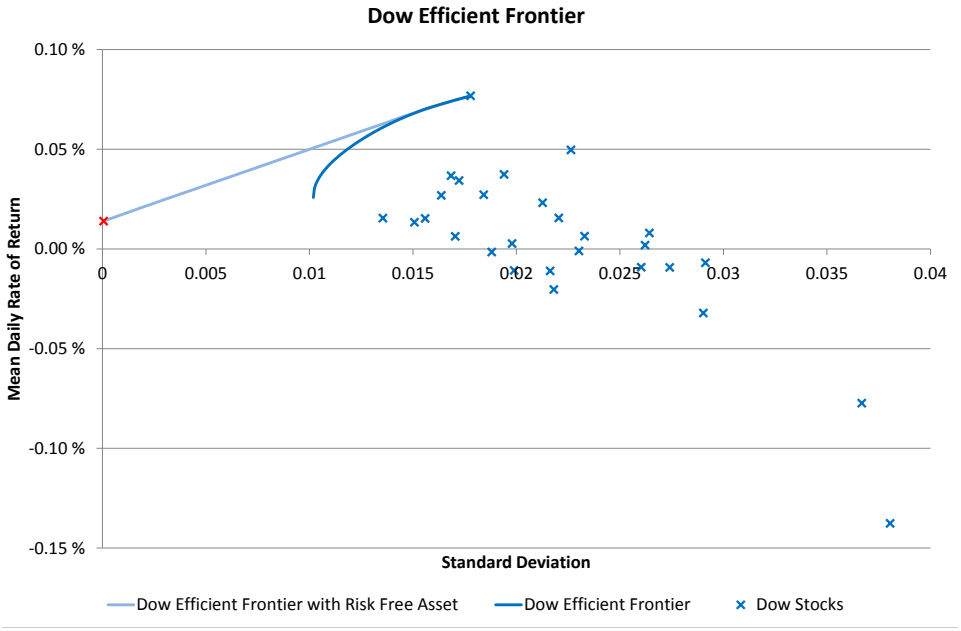


Figure 2.4: The chart shows the efficient frontiers of the Dow Jones Industrial Index using daily sampling from 2000 to 2012. Risk free asset is 5 year US Treasury bond.

3. Strong EMH additionally claims that prices instantly reflect even hidden or "insider" information.

The efficient market hypothesis was widely accepted until the 1990s when behavioral economics started to gained ground. Empirical analysis has consistently found faults with EHM [20; 27; 112]. In spite of consistent substantial inefficiency and controversy it is still widely considered a valid starting point [28]. The EHM laid the foundation for modern portfolio theory.

2.1.4 Portfolio Performance Measurements

As discussed in the previous section, portfolio performance is generally measured as the achieved return relative to the risk taken. The classical approach as used in modern portfolio theory is to use mean return as the (expected) return measure and variance as the risk measure. It is not given that these are the best metrics for measuring performance. In this section we review some of the alternatives.

Cumulative Return

Cumulative return R_T, not to be confused with the rate of return matrix $\mathbf{R}$, is the return achieved over some period of time T. It can be calculated in two ways. The

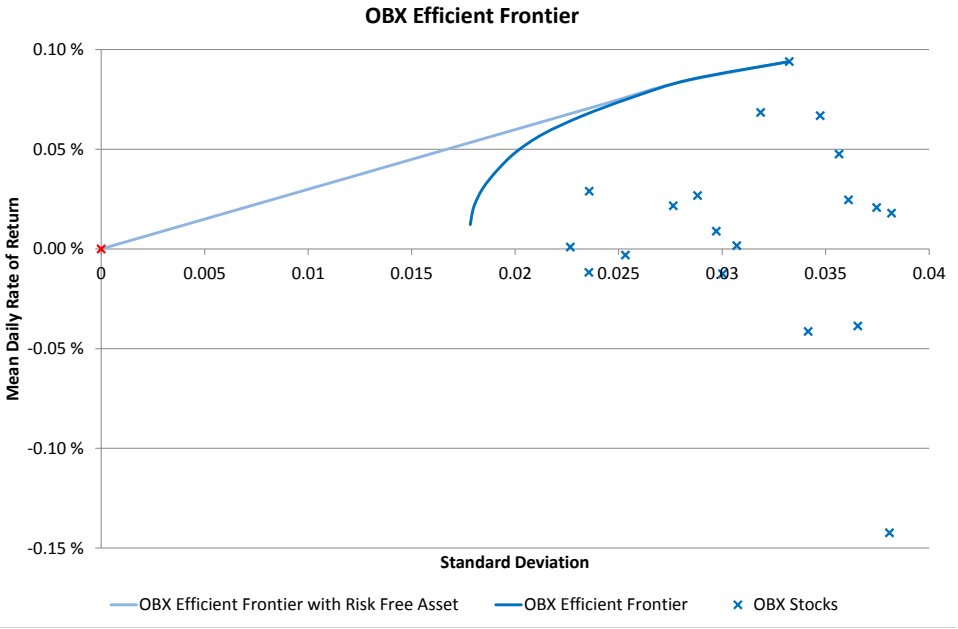


Figure 2.5: The chart shows the efficient frontiers of the 25 most liquid stocks of Oslo Stock Exchange (OBX) using daily sampling from 2006 to 2012. Risk free asset is cash in NOK.

first way to calculate the cumulative return uses the compounded interests method, defined by (2.13). This can be extended to (2.14) which is a useful definition for market simulation purposes.

$$R_T = \left(\prod_{t=0}^{T} 1 + r_t \right) - 1 \tag{2.13}$$

$$R_t = (1 + R_{t-1})(1 + r_t) - 1 \tag{2.14}$$

The alternative uses the price p_t at the start of the period $t = 0$ and the end $t = T$, this version is shown in figure (2.15)

$$R_T = \frac{p_T - p_0}{p_0} \tag{2.15}$$

Cumulative return is a practical measure that reflects what investors in the real world are looking for. It is the actual return on investment that a trader can attain by applying a specific portfolio or trading strategy. To a degree it incorporates an inherent measure of risk due to the long time horizon.

Table 2.2: Trader Strategy Performance Measures

Feature	Description
Mean Return	The simple mean of a sample of returns
Return Variance	The simple variance of a sample of returns
Cumulative Return	The return on investment over time
Jensen's Alpha	Abnormal return based on CAPM
Beta	The relative variance of a target portfolio relative to the market
VaR	A probabilistic risk measure that takes into account downside risk
Omega	A probabilistic risk measure that takes into account upside and downside risk
Traynor	Risk return ratio with beta as risk measure
Sharpe	Risk return ratio with standard deviation as risk measure
Sortino	Risk return ratio with standard deviation of losses as risk measure

Alpha and Beta

The alpha and beta of a portfolio or asset are measures of abnormal return and relative volatility. The alpha says how much more return the investment object is generating compared to a benchmark, while the beta says how much volatility, i.e., risk, it is incurring relative to the benchmark. There are two main ways of calculating the measures, one uses explicit calculations the other linear regression.

Capital Asset Pricing Model. William F. Sharpe[131] devised the Capital Asset Pricing Model (CAPM) as a theoretically appropriate rate of return of an asset or portfolio given the assumptions of MPT. The beta is a key parameter of CAPM.

The beta $\beta_{\boldsymbol{w}}$ of a target portfolio $\boldsymbol{w}$ is a measure of volatility compared to a benchmark portfolio $\boldsymbol{m}$ typically using a weight configuration representing the market. The Beta, as defined by Sharpe[130], is computed by the covariance between the target portfolio returns and the benchmark portfolio returns, divided by the benchmark return variance. This is expressed in (2.16) where $\boldsymbol{r}_{\boldsymbol{w}} = \mathbf{R}\boldsymbol{w}$ is the time series returns of the target portfolio $\boldsymbol{w}$ applied to the return matrix $\mathbf{R}$,

and the benchmark portfolio $\boldsymbol{w}$ yielding returns $\boldsymbol{r_m} = \mathbf{R}\boldsymbol{m}$.

$$\beta_{\boldsymbol{w}} = \frac{\text{cov}\left(\boldsymbol{r_w}, \boldsymbol{r_m}\right)}{\text{var}\left(\boldsymbol{r_m}\right)} \tag{2.16}$$

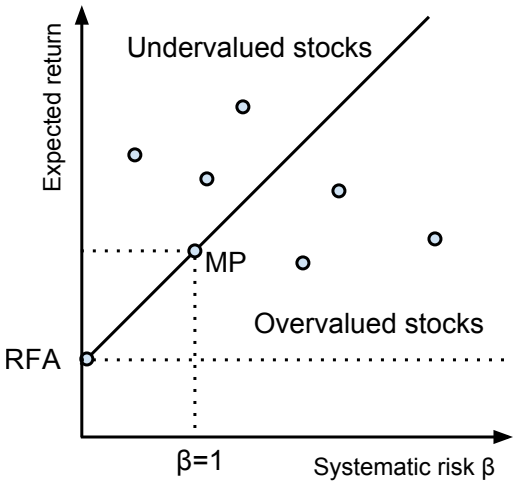


Figure 2.6: The security market line shows ...

A general version of CAPM is defined by (2.17), where $E(\boldsymbol{r_w})$ is the expected return of a portfolio $\boldsymbol{w}$, r_f is a risk-free rate and $E(\boldsymbol{r_m})$ is the expected return of the market. In practical terms the expected return of a portfolio can be calculated by using $E(\boldsymbol{r_m}) = \bar{r}_{\boldsymbol{m}}$, $r_f = \bar{r}_f$ and $\beta_{\boldsymbol{w}}$ as defined in (2.16).

$$E(\boldsymbol{r_w}) = r_f + \beta_{\boldsymbol{w}}(E(\boldsymbol{r_m}) - r_f) \tag{2.17}$$

By also calculating an explicit estimate of $E(\boldsymbol{r_w})$ in (2.17), say $E(\boldsymbol{r_w}) = \bar{r}_{\boldsymbol{w}}$, the discrepancy that arises can be defined as the alpha $\alpha_{\boldsymbol{w}}$ of a the portfolio $\boldsymbol{w}$ relative to some benchmark portfolio $\boldsymbol{m}$. The alpha is often termed abnormal return, and used as a performance measure that indicates if a portfolio or trading strategy outperforms the market given some risk free rate. A version of this alpha is Jensen's alpha defined by (2.18). An interpretation of alpha is that it measures how much of a target portfolio's return is attributable to the portfolio generator, adjusted for market risk. This makes it a good measure of how well the portfolio as a whole has performed historically, while incorporating risk.

$$\alpha_{\boldsymbol{w}} = \bar{r}_{\boldsymbol{w}} - \bar{r}_f + \beta_{\boldsymbol{w}}(\bar{r}_{\boldsymbol{m}} - \bar{r}_f) \tag{2.18}$$

Estimating Alpha and Beta by Regression. The alpha $\alpha_{\boldsymbol{w}}$ and beta $\beta_{\boldsymbol{w}}$ of a target portfolio $\boldsymbol{w}$ can also be computed by linear regression analysis of the excess returns $\boldsymbol{r_w} - \boldsymbol{r}_f$ of the target portfolio versus the excess returns $\boldsymbol{r_m} - \boldsymbol{r}_f$ of a benchmark portfolio $\boldsymbol{m}$. Here $\boldsymbol{r}_f$ is a time series of risk free rate of returns, it can sometimes be exchanged by a scalar risk free rate r_f. Linear regression is performed by minimize the error terms $\boldsymbol{\varepsilon_w}$ of (2.19) with regard to parameters $\alpha_{\boldsymbol{w}}$ and $\beta_{\boldsymbol{w}}$ yielding the **security characteristic line** (SCL) of (2.20) for any time t. The beta $\beta_{\boldsymbol{w}}$ is the slope of the line while the alpha $\alpha_{\boldsymbol{w}}$ is the intercept.

$$\boldsymbol{r_w} - \boldsymbol{r}_f = \alpha_{\boldsymbol{w}} + \beta_{\boldsymbol{w}}(\boldsymbol{r_m} - \boldsymbol{r}_f) + \boldsymbol{\varepsilon_w} \tag{2.19}$$

$$r_{t\boldsymbol{w}} - r_{tf} = \alpha_{\boldsymbol{w}} + \beta_{\boldsymbol{w}}(r_{t\boldsymbol{m}} - r_{tf}) + \varepsilon_{t\boldsymbol{w}} \tag{2.20}$$

Measuring Risk Using Probability Distributions

Variance as a measure of risk has been criticized for treating profit and loss equally. In response to this there have been developed alternative risk measures which try to measure risk in a way that treats upside and downside risk differently. **Value at Risk** (VaR) is one such measure measure[89], while the **Omega ratio** is an improved concept that expands on the VaR probabilistic approach.

Value at Risk. The VaR value represent the most one can expect to lose within a given confidence interval over a given time period. It may say something like, "With a 99% probability, we will not lose more than 4% in the next week." Additionally, VaR can be used as the risk measure to calculate the efficient frontier[69].

Given the stochastic rate of return r of some investment object, and a probability target p we can define the value at risk r_{VaR} to be the return threshold that fulfills equation (2.21). Here $F_{\boldsymbol{w}}(r)$ is the cumulative distribution function and $f_{\boldsymbol{w}}(r)$ is the probability density function of the return for applying portfolio $\boldsymbol{w}$.

$$1 - p = \Pr(r \leq r_{\text{VaR}}) = F_{\boldsymbol{w}}(r_{\text{VaR}}) = \int_{-\infty}^{r_{\text{VaR}}} f_{\boldsymbol{w}}(r)\,\mathrm{d}r \tag{2.21}$$

In practical terms, we can find r_{VaR} by using the cumulative probability distribution of a sample of returns. That is, if we have T sample returns r_t in a time series vector $\boldsymbol{r}$ we can find r_{VaR} by sorting $\boldsymbol{r}$ in ascending order and selecting $r_{\text{VaR}} = r_k$ where $k = \lfloor (1-p)T \rfloor$. This method is how figure 2.7 was created.

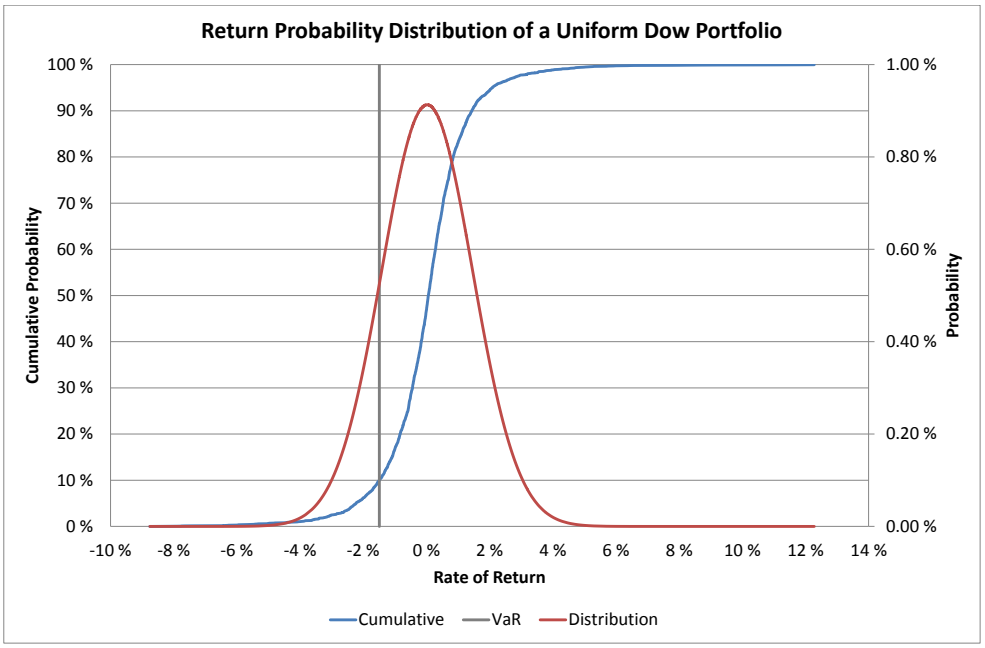


Figure 2.7: The chart shows the cumulative (left axis) and non-cumulative (right axis) probability distribution of a uniform buy-and-hold portfolio consisting of the Dow Jones Index companies. The vertical line shows a one-day 90% VaR, i.e., there is a 10% probability of a 1.5% or worse loss within the next day.

Omega Ratio. In 2002 Keating and Shadwick published their paper criticizing the use of mean and variance and proposing their own performance measure, the omega[92]. The problem with all measures based on mean and variance is that they cannot capture all interesting risk and return features unless the returns are normally distribution[92]. The omega ratio defined in 2.22 is the area under the curve of the cumulative distribution function $F(r)$ up to the threshold r_g divided by the area over the curve of $F(r)$ from the threshold r_g and on.

$$\frac{\int_{r_g}^{\infty}(1-F(r))\mathrm{d}r}{\int_{-\infty}^{r_g}F(r)\mathrm{d}r} \tag{2.22}$$

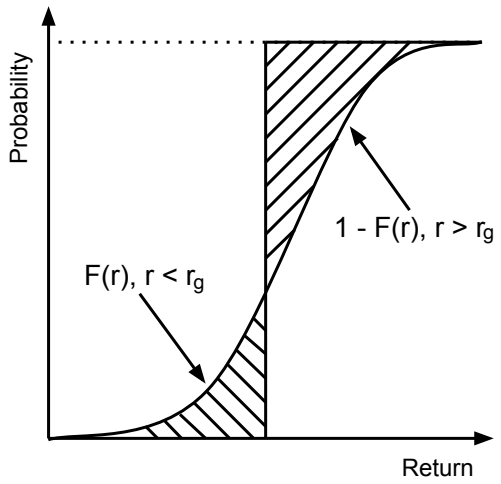


Figure 2.8: The omega ratio is calculated by using different areas of the cumulative distribution function $F(r)$.

Risk-Return Ratios

As discussed in section 2.1.3, utility functions of different kinds are useful to combining both the return and risk aspect of investing. One class of these utility functions are the **risk-return** ratios.

Treynor Ratio. Treynor was the first to create such a measure, and suggested splitting risk in market risk and individual security risk. He introduced the security market line which defines the relationship between the portfolio returns and the market rates of return. The volatility between portfolio returns and market returns is represented by the beta[67]. The Treynor ratio is defined by 2.23, where $r_{\boldsymbol{w}}$ is the return of the portfolio $\boldsymbol{w}$, r_f is the risk-free rate and $\beta_{\boldsymbol{w}}$ is the beta.

$$\frac{E(\boldsymbol{r}_{\boldsymbol{w}}) - r_f}{\beta_{\boldsymbol{w}}} = \frac{\bar{r}_{\boldsymbol{w}} - r_f}{\beta_{\boldsymbol{w}}} \tag{2.23}$$

Sharpe Ratio. William F. Sharpe introduced the Sharpe ratio in 1966[132], this is close to identical to Treyor with the beta switched for the standard deviations. The Sharpe ratio thus incorporates all risk, not just the systematic one. Given constant r_f risk-free rate the Sharpe ratio is defined by (2.24). An updated version that accounts for variable risk-free rate can be defined by (2.25). This ratio is closely linked to his CAPM for which he won a Nobel prize together with Markowitz[2].

$$\frac{E(\boldsymbol{r}_{\boldsymbol{w}}) - r_f}{\sqrt{\text{var}\,(\boldsymbol{r}_{\boldsymbol{w}} - r_f)}} = \frac{\bar{r}_{\boldsymbol{w}} - r_f}{\sigma_{\boldsymbol{w}}} \tag{2.24}$$

$$\frac{E(\boldsymbol{r}_{\boldsymbol{w}} - \boldsymbol{r}_f)}{\sqrt{\text{var}\,(\boldsymbol{r}_{\boldsymbol{w}} - \boldsymbol{r}_f)}} = \frac{\frac{1}{T}(\boldsymbol{r}_{\boldsymbol{w}} - \boldsymbol{r}_f)^{\mathrm{T}}\mathbf{1}}{\sqrt{\text{var}\,(\boldsymbol{r}_{\boldsymbol{w}} - \boldsymbol{r}_f)}} \tag{2.25}$$

Sortino Ratio. As with Omega and VaR, the Sortino Ratio accounts for downside risk. Frank A. Sortino modified the Sharpe ratio to account only for downside risk in his Sortino ratio[65], defined by (2.26). Here r_g is a return target which typically is selected to be $r_g \geq r_f$. $f_{\boldsymbol{w}}(x)$ is the probability density function of the returns of portfolio $\boldsymbol{w}$

$$\frac{E(\boldsymbol{r}_{\boldsymbol{w}}) - r_g}{\sigma_{DR}} = \frac{\bar{r}_{\boldsymbol{w}} - r_g}{\sqrt{\int_{-\infty}^{r_g}(r - r_g)^2 f_{\boldsymbol{w}}(r)\ \mathrm{d}r)}} \tag{2.26}$$

In practice the downside risk can be computed by calculating the variance of the entries of $\boldsymbol{r}_{\boldsymbol{w}}$ that are less than the target return r_g. Notice that we are not talking about setting the entries that don't fulfill the requirement to zero, but actually removing them. In other words, let $\boldsymbol{r}_{\leq r_g, \boldsymbol{w}}$ be a vector of the entries $r_{t\boldsymbol{w}}$ of $\boldsymbol{r}_{\boldsymbol{w}}$ such that $r_{t\boldsymbol{w}} \leq r_g$, then the downside risk is defined by (2.27), using a constant target r_g.

$$\sigma_{DR}^2 = \text{var}\left(\boldsymbol{r}_{\leq r_g, \boldsymbol{w}} - r_g\right) = \text{var}\left(\boldsymbol{r}_{\leq r_g, \boldsymbol{w}}\right) \tag{2.27}$$

If the target is a time series of floating rate of returns $\boldsymbol{r}_g$, e.g., national bonds, the last simplification cannot be performed. In this case we must use the more general (2.28).

$$\sigma_{DR}^2 = \text{var}\left(\boldsymbol{r}_{\leq r_g, \boldsymbol{w}} - \boldsymbol{r}_g\right) \tag{2.28}$$

The Sortino ratio can be thought of as the excess of the investor's return per unit of downside risk, or overperformance divided by the root-mean-square of underperformance. For an example of the critique on the use mean and variance see figure 2.9.

2.1.5 Behavioral Finance

Behavioral Finance is a part of the field behavioral economics where cognitive and emotional factors are used to explain economic decisions of actors. This field is concerned with bounds of rationality of economic agents. These behavioral models typically integrate psychology and neo-classical economic theory. In 1985 De

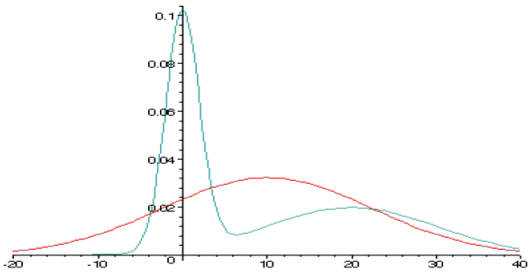


Figure 2.9: These distributions have the same mean and variance. One can question whether they represent the same risk profile. This is the issue the omega ratio is meant to solve. The Sortino ratio also goes some way to mitigate this issue by only considering downside deviations.

Bondt and Thaler published their result showing that people overreact to unexpected or dramatic news events. Thus discovering substantial weak form market inefficiency, this result was later reproduced and is seen by some of the start of behavioral economics [33]. This effect is further confirmed by Kahneman & Tversky well known article about the human bias in decision making under risk. They criticize the expected utility theory model and propose their own prospect theory. These prospects show several pervasive effects inconsistent with the expected utility theory. One effect prospects exhibit are to underweight outcomes that are merely probable compared to certain ones. This contributes to risk aversion in choices involving sure gains and to risk seeking in choices involving sure losses[91].

Proponents of the EMH have criticized the behavioral economics for just being a collection of anomalies. Most prominent is Fama, the father of EMH. He claims EMH is a model and as all models it does not perfectly explain the real world. He still contend that it is the best basis for investors entering the market as there are few empirical studies that have managed to capitalize on the anomalies they have found in any significant degree [56]. Others contend that experimentally observed behavior of the kind Kahneman et al. did has little effect on the market situation as learning opportunities and competition will ensure at least close to rational behavior [111].

2.1.6 Financial Rationale

This section argues for why the system will reach its goals from a financial theoretical perspective.

The purpose of the system is to identify and purchase assets that are underpriced and generate return by selling after the price corrects. According EMH this ought to be impossible as all information should already be reflected in the price, thus foil-

ing any arbitrage opportunities. However, if one looks to behavioral finance they argue that the actors in the market often overreact or make other irrational decisions [91]. They argue that markets may exhibit irrational exuberance emerging from negative or positive feedback loops. This creates opportunities for exploitation[4] of such behavior and thus generate return by correcting the EMH anomalies. Even Eugene Fama, the father of EMH, has admitted to the fact that there are anomalies in the efficient market model, but retorts that it is only a model, yet a model which, to this day, is the best representation of financial markets there is[56].

According to findings of Bondt and Thaler [33] many actors in financial markets behave irrational. This is one of the foundations of behavioural finance. These results have later been confirmed, and several more similar EMH anomalies have been found. Banz[146] and Reinganum [118] found that firms with low market values of equity was consistently undervalued and vice versa, this is termed the **small firm effect** (SFE). Lamoureux and Sanger [95] showed this same effect on a different market, thus concluding that the effect is independent of market structure and further strengthening the result. This, however, has been challenged by Keim [19] and Brown et al., [37] who found instabilities in SFE. Their results indicate that other factors than posed in SFE can affect stock and the overall market return. This suggests that alternative factors should be incorporated in expected return models. [66].

Other studies have found that the risk-return relationships may be captured by other combinations of company and market specific data. Basu (1983) finds E/P (earnings/price) and firm size both to be constituents of average returns for NYSE/AMEX listed firms[66]. In addition to the psychological arguments against the traditional finance there is the so called "limits to arbitrage" argument.

Some argue that underperforming actors will gradually be eliminated as they will lose money through their irrational investments[56]. This is countered by the second foundational block of behavioral finance, the "limits of arbitrage" argument. The argument says that it can be difficult for rational actors to undo dislocations caused by less rational traders[25]. This indicates that there exist latent arbitrage possibilities in a market at any time. The plan is to create a system which is able to exploit these opportunities. This will thus contributing to moving the market towards a more rational and efficient equilibrium, while achieving a profit premium above the average market.

Supported by the behavioral finance arguments the system will use historical data

[4]Here the term "exploitation" is used in a game theoretical sense. That is, changing ones strategy based on another's poorer strategy.

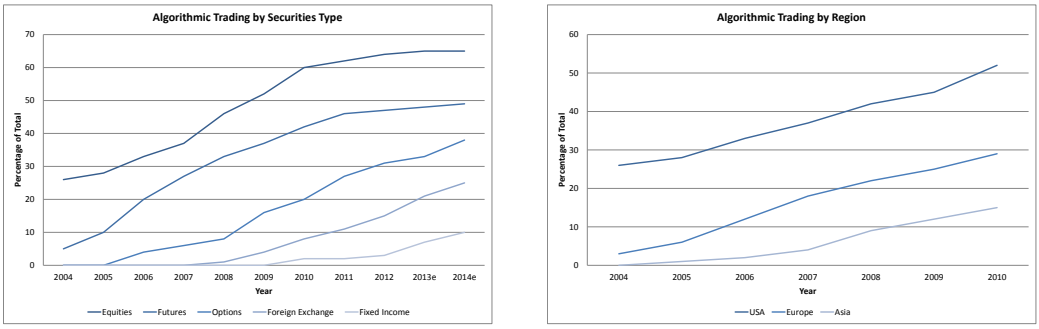
as the basis for trading. It will have none of the cognitive biases which human traders tend to display[91]. Based on the literary research, there seems to be a possibility of identifying arbitrage opportunities. If identified, the system will use those findings to make trading decisions. However, potential issues may arise from two sources:

1. It might not be possible to base trading on historical data. Though, multiple anomalies like SFE and the January effect seems to confirm that it is indeed possible[79; 80; 141].
2. The system may not be able to adequately identify latent arbitrage opportunities.

2.1.7 Automated trading

Automated trading started with the use of computers to execute large orders. This often included algorithms for splitting up orders with the aim to achieve a higher average price. Later this has evolved to algorithms using strategies to suggest actions or trade on their own from. In recent time the aforementioned high-frequency trading has bloomed and even large banks like Credit Suisse and Deutsche Bank has entered with their own algorithms.

High speed traders now dominate most markets in terms of bid and ask orders. Algorithmic algorithms accounts for over 60% of all trading on stock markets in the world as of 2011 see figure 2.10a. The trend is most dominating in USA, but Europe and Asia are following, see figure 2.10b.



(a) Algorithmic trading in the world as % of their respective total [6]

(b) Algorithmic trading by geography as % of total [6]

The debate whether HFT improves or impedes market efficiency is still open. Many claim to show that it reduces the spread and improves liquidity [35; 39; 77; 82; 83],

while others claim they manipulate prices and increase spreads [85]. This is still an open debate, though there seems to be more evidence that HFT and other algorithmic trading improve the overall quality of the markets. Even some that conclude HFT is the natural next step in the evolution of the trading process still advocates changes in mechanism to reduce the risk for market manipulation or market breakdowns[55].

Algorithmic algorithms have been accused of being partly responsible in several market crashes, as far back as to the "Black Friday" in 1987 to the far more recent "Flash crash" in 2010. Such incidents and the growing influence on the market by these algorithms have led to many lobbying for more strict regulations or even forbidding large areas of this practice altogether. There is little chance of a ban on the practice, but restrictions on speed or regulation of the frequency and or volume of trades might be introduced.

ESMA, the European Securities and Markets Authority published on the 24th of February 2012 the "Systems and controls in an automated trading environment for trading platforms, investment firms and competent authorities" guidelines. These are meant to ensure stable electronic market systems in Europe. The guidelines require surveillance of the algorithms behavior and further testing to avoid erroneous behavior.

Oslo Stock Exchange is planning to introduce a fee on placing many orders without executing them[22]. This will occur after a set number of orders are placed without making an actual trade[7]. Such a fee could severely hamper most HFT systems that rely on posting vast amount of orders.

Legal. An issue in automated trading is breaking the laws of trading, whether deliberate or not. Market manipulation, deliberate attempts to interfere with the fair operations of the market is illegal in most financial markets. The keyword here is deliberate and when a computer's action is deemed to be deliberate. An algorithm with non-deterministic outcome might commit actions that could be seen as effectively market manipulation. However as long as this is a random outcome it is not deliberate and can thus not be classified as illegal market manipulation. Another interesting aspect is other actors that might exploit weaknesses in a running system. A simple system can have deficiencies that another actor can capitalize on. In this case the exploited actor is in effect behaving irrationally and causing inefficiencies and the exploiter is only correcting such mistakes. Such a case is at the time ongoing in the Norwegian justice system, and in the last verdict the exploiters were deemed innocent of any crime. The final outcome of this case will be important for the future of automated trading. And the first judgment in favor of the malfunctioning bot indicates that the justice system is immature in this

aspect and a final judgment in their favor would send signals that any automated system can have a free lunch and then sue any possible loss on the account of exploitations.

Effects of algorithmic trading. Trading sizes is reduced; many former human jobs are now done by computers. The connection speed has increasing importance due to low-latency trading. There is more competition between exchanges and the more fully automated ones like NASDAQ have acquired a larger market share. Such competition have led to faster processing systems at exchanges, in June 2007 London Stock Exchange launched a new system promising 10 ms turnaround time from order placement to confirmation.

2.2 Data Features

This section elaborates on the data the trading systems take as input. Regardless of how well the trading systems operate they require quality data to produce any valuable input. No matter the computation power, if garbage comes in, garbage comes out. The input data is divided into three categories.

1. Technical
2. Fundamental
3. Context

2.2.1 Technical Data

Technical data is data directly derived from the market itself, typically variations on price and volume. An inherent assumption in technical analysis is thus that prices can be predicted with historical data. This is a breach with MPT and even weak form of market efficiency[59]. While most academics are skeptical of technical analysis many practitioners still employ it as an integral part of their analysis [129; 138]. Taylor and Allen additionally found that the use of technical analysis was reversely correlated to the length of the time horizon in question[138]. Irwin and Park found in 2003 that 56 of 95 modern studies showed financial analysis to produce positive results[116]. Several of these were however rendered dubious by issues like data snooping[5]. Technical analysis is still largely considered to be pseudoscience by academia [117].

Technical analysis is generally divided into quantitative analysis and charting.

[5]Data snooping is the inappropriate use of data mining to uncover misleading relationships in data

Quantitative analysis is basically just statistical analysis with market data. Technical data and especially quantitative financial data is ideally suited for algorithmic analysis due to the computational power required. Charting, on the other hand, tries to apply visual representations and rules of thumb in order to find repeating patterns. In a sense, it is applying human intuition and hunches. Since this is outside the realm of scientific discourse this part of technical analysis will be avoided.

The following are some quantitative methods that are applied in technical analysis. Note that some of these are measures that are used by both wings of technical analysis, and can thus be mildly controversial to use.

For all following indicators $\boldsymbol{p}$ is all data of a single time series (with only one feature), and p_t is the latest, i.e., current, value of the time series. In most cases they represent price directly, hens the name. Notice that even though the entire history of the time series $\boldsymbol{p}$ is supplied to some of the indicators not all of it is used.

Table 2.3: Technical Financial Indicators

Indicator	Full Name	Reference
SMA	Simple Moving Average	[45]
EMA	Exponential Moving Average	[121]
RSI	Relative strength index	[10]
MACD	Moving average convergence/divergence	[16]

Simple Moving Average

The simple moving average (SMA) is the rolling mean of a time series. A window of size τ at period t yields the definition of (2.29). It is possible to apply some weights to each term of the denominator, making it a weighted moving average (WMA). This can be used to emphasize recent data, more than old data.

$$\mathrm{SMA}(\boldsymbol{p}; \tau) = \frac{p_t + \cdots + p_{t-\tau}}{\tau} \tag{2.29}$$

Exponential Moving Average

Arguably a better way to emphasize new data is the exponential moving average (EMA). It is an online moving average that is recursively updated. It weights new data with $1/\tau$ and old data, i.e., the previous EMA, with the remainder $1 - 1/\tau = (\tau - 1)/\tau$. The parameter τ is some times replaced by some variable

α such that $\alpha = 1/\tau$, the reason for using τ in the first place is the intuition of applying a integer window length as with other moving averages, although this is not required. The exponential moving average has the nice property of gradually dwindling the importance of old data, yet never entirely removing its effect.

$$\mathrm{EMA}(p_t;\tau) = \frac{1}{\tau}p_t + \frac{\tau-1}{\tau}\mathrm{EMA}(p_{t-1};\tau) \tag{2.30}$$

Relative Strength Index

The relative strength index (RSI) is a momentum oscillator that measures the velocity and magnitude of the movements of a time series. It uses the relative strength (RS) defined by (2.31) where $I(\cdot) \in \{0,1\}$ is the indicator function such that $I(\text{true}) = 1$ and $I(\text{false}) = 0$;

$$\mathrm{RS}(p_t;\tau_U,\tau_D) = \frac{\mathrm{EMA}(p_t I(p_t > 0);\tau_U)}{\mathrm{EMA}(-p_t I(p_t < 0);\tau_D)} \tag{2.31}$$

This results in the RSI definition of (2.32) which spans the interval $[0,1]$.

$$\mathrm{RSI}(p_t;\tau_U,\tau_D) = 1 - \frac{1}{1+\mathrm{RS}(p_t;\tau_U,\tau_D)} \tag{2.32}$$

Moving Average Convergence/Divergence

The moving average convergence/divergence (MACD) is an indicator that utilizes a set of nested exponential moving averages. The **MACD line** is a the difference between a fast EMA and slow EMA. The MACD signal line is another EMA of the MACD line. The final **MACD histogram** is the difference of the MACD line and the MACD signal line. It oscillates around the "zero" line having no particular limit on its real numbered codomain. The calculation of the MACD lines is sown in (2.33). The MACD histogram is used as the final indicator of the later trading system caring its name.

$$\begin{aligned}
\mathrm{MACD}_{\mathrm{line}}(p_t;\tau_F,\tau_S) &= \mathrm{EMA}(p_t;\tau_F) - \mathrm{EMA}(p_t;\tau_S) \\
\mathrm{MACD}_{\mathrm{sgn}}(p_t;\tau_F,\tau_S,\tau_{\mathrm{sgn}}) &= \mathrm{EMA}(\mathrm{MACD}_{\mathrm{line}}(p_t;\tau_F,\tau_S);\tau_{\mathrm{sgn}}) \\
\mathrm{MACD}_{\mathrm{hist}}(p_t;\tau_F,\tau_S) &= \mathrm{MACD}_{\mathrm{line}}(p_t;\tau_F,\tau_S) \\
&- \mathrm{MACD}_{\mathrm{sgn}}(p_t;\tau_F,\tau_S,\tau_{\mathrm{sgn}})
\end{aligned} \tag{2.33}$$

2.2.2 Fundamental Data

Fundamental data is financial and accounting data related to the asset in question. The financial data is used to forecast the price by identifying whether the asset seems over- or underpriced by the market. Fundamental trading are more based on qualitatively analysis than technical and though certain ratio metrics are common a deeper understanding of what the asset represents are often required to do a good analysis. For this reason computers are less suitable at fundamental than technical analysis. Common quantitatively fundamental data are ratios built by accounting data, e.g. the price/earnings ratio (P/E), other more qualitatively data can also be used e.g. news stories. Fundamental analysis also breach with MPT and the semi-strong form of market efficiency[59].

Fundamental analysis is a much more accepted practice. It is considered the conventional analysis technique as it allows one to make decisions based on a personal valuation rather than the market [73]. Several papers find underlying relations between fundamental data and asset prices [8; 9; 97; 113]. Aside from being hard to analyze automatically fundamental data generally have a much slower update frequency as most data are linked to quarterly report. Combined with the time it takes to do a thorough fundamental analysis this explains why fundamental trading normally is most prominent in long term investment decisions. In algorithmic trading fundamental data can be very useful as context or classification data and then combined with technical data for its abundance and frequency.

2.2.3 Context Data

Context data in this paper is a general term used for data that is common to the whole market in question and deemed potentially relevant to prices. Some examples are bond yields, index values, unemployment rates, and other asset prices like oil or even simple information as which month of the year it is.

One of the central aims of the trading systems in this paper is to be able to identify and exploit anomalies in the market. The method used in this system train on historical data. In order to be able to identify upcoming situations that match previous learned anomalies the systems need an understanding of the market and a context to learn situations in. Most trading systems are centered on ANNs and the context data is used to give the ANN a context relative to the other data and can as such learn to recognize certain situations and exploit patterns. Without a context it is much harder for such techniques to discover patterns. Macro-variables have been found to further improves the predictive value of fundamental data [97]. There are four different categories of context data at use in these systems.

1. Time

Table 2.4: Fundamental Financial Indicators

Feature	Description	Reference
Price/Earnings	Compares stock price to company earning. Gives indication of over-/under valuation	[43; 50]
Price/Book	Compares market stock price with the accounted price, book value. Another indication of over-/under valuation	[43]
PEG	Compares P/E to the company' annual EPS growth. Estimates value while accounting for growth	[50]
Price/Sales	Gives an indication of recent performance related to the stock price	
Market Cap	The total market capitalization value of the asset	
Enterprise Value	Market Cap + Debt + Minority Interests + Preferred Shares - Cash - Cash Equivalents	
EBITDA	Earnings before interests, tax, depreciation and amortization	[119]
EPS	Measures earnings in relation to number of shares.	[42]
Forcasted EPS	Forecasts for EPS done by human analysts	[12]
5Y Net Growth	Gives an indication of development. Can together with dividends help distinguish between cash cows and growth companies	
Dividend Yield	How much the company pays in dividends. Have been found to have predictive effect on stock returns	[60; 104]

2. Other Assets
3. Macro Data
4. Indices

Several time dependent patterns have previously been found in stock markets. Most common are the January effect[33] and the weekend anomaly, where there is a bias toward strong performance on Fridays compared with Mondays [68; 134]. Other assets like copper have also been shown to have predictive effect for longer term business cycles [4; 36] and oil is a good indicator of world events and have been shown to predict stock prices [48]. Macro-economic conditions give a good economical context and gives insight into long term trends. Studies have shown macro-economic data to be important factors in determining the possibility of financial distress [143]. Stock indices from important regions around the globe are

Table 2.5: Context Data

Feature Extractor	Description
Periodic Time Indicators	Cyclical time counters that tell the system what period it is in. Possible indicators are the month of year, day of month, day of week and trading hour.
Bonds	The yield of US treasury bonds are used as an indication of the risk free rate.
Market Indices	Stock market indices like Dow or S&P500 gives good indications of the general trend in the markets
Commodity Prices	Commodity prices gives a good indication of the economic state in the world and are often correlated with stock markets
Macro-economic Indicators	Macro-economic indicators like unemployment rate and GDP gives good indication of market prospects

used to find global patterns and to provide a relative weight to the magnitude of the local price movements. The list of the actual data used follows in the next section.

Market Performance Measurements

Another set of indicators which may be of interest are macro indicators and period indicators. These are factors that are not directly related to any single security, but still considered to affect the stock market indirectly. Many of the indicators may seem trivial, but it is important to include factors like time data, such that the system can use it to identify patterns. In table 2.5 the general categories of indicators which can be used have been listed. Together these are meant to provide an important context to the fundamental and technical indicators as well as providing a general measure of the state of the market altogether.

2.2.4 Asset Data

The price data received from the market includes more than just one price for each stock. Here is the different inputs used listed according to category.

Table 2.6: Asset Data

Feature	Description	Dataset	Reference
High	The highest price of the day	OBX & Dow	
Low	The lowest price of the day	OBX & Dow	
Ask	The price at The lowest price a seller is willing to sell at	OBX	
Bid	The highest price a buyer is willing to buy at	OBX	
Open	The price the market opened at	Dow	
Close	The price the market closed at	OBX & Dow	
Volume	The number of shares traded in a security during a given period of time. Evidence suggest a negative relationship between volume and future returns	OBX & Dow	[105]
Adjusted Close	The close price adjusted for any distributions and corporate actions that occurred at any time prior to the next day's open	Dow	

2.3 Technology Methods

2.3.1 Computer Science Concepts

This project is in large parts a computer science project with most of the workload connected to the development of a large framework requiring a structures system development approach. In this section a few important concepts in this regard are presented.

Object Oriented Programming

There are many **programming paradigms** that are suitable to perform heavy computations. General purpose procedural programming languages like C and assembly can compute individual operations very fast, but have limited pre-existing internal structure. Other languages like Matlab and R are specialized in mathematical and statistical computations, but these are somewhat lacking when creating large complex systems. Object oriented programming, available in programming languages like C++, C# and Java, is a programming parading that might be slower in many respects, but has the advantage of flexible data structures. Sim-

pler and faster implementation of more advanced data structures and software design patterns reduces development time and aids in test driven development. It also allows for a more experimental approach to development since its modular nature makes it possible to radically alter one part of the system without **redesign**, that is rewriting the whole system.

Parallel Processing and Threading

A major issue with processing vast amounts of data is the need for **parallel processing**. A naive implementation of computational intensive tasks may take many orders of magnitude longer than an efficient implementation. This can be mitigated by using good algorithms, but also executing those good algorithms in parallel can greatly increase efficiency. Parallel processing can be done either on a single computer as a "miniature program" called a **thread** or on multiple networked computers as **client programs**. The common denominator is the need to create encapsulated objects that may process separately, combining their work only intermittently. This makes object oriented programming the ideal programming paradigm for parallel processing, since objects are naturally modular and, if done right, encapsulated and threadable. This puts requirements on how a system is implemented and can reduce performance on simple computational tasks, though increase it on more complex larger task. Many design patterns aid in increasing parallelization, the producer-consumer pattern and Kahn processing networks are among them.

Producer-Consumer Problem

Design patterns are common solutions to common problems in software development. These help to improve both the credibility and the readability of the code and works as best practices in the industry.

A classical design pattern within data processing is the producer-consumer problem also known as the "bounded-buffer problem". This is a classic example of a multi-process synchronization problem. One process produces something that another is set to consume. The producer has to store the products somewhere for the consumer to find them and for intermediary storage before the consumer is ready. This is typically called a buffer. Here a problem is that the buffer might be full, and the solution is to set the producer to sleep while the buffer is full. In such a way the buffer has to notify the producer when it is not full again. In the same way the consumer should go to sleep in the event of an empty and be notified by the buffer when new products arrive. The solution is commonly reached by using semaphores. In Java this can be achieved by the use of a shared BlockingQueue as buffer.

Kahn Process Networks

In signal processing a Kahn process network (KPN) is a useful design pattern that allows for parallel and even distributed processing of input signals. In principle KPNs are interconnected processes that form the nodes of a directed acyclic graph (DAG). Processes are often implemented as self-sufficient threads locally or clients if distributed. They may employ a producer-consumer design pattern in order to pass processed data downstream. In mathematical terms, a process can be viewed as real-valued function f with n inputs and k outputs, that is $f : \mathbb{R}^n \rightarrow \mathbb{R}^k$. Collectively these KPN functions act as one big composite function that transform input to some desired output. Figure 2.10 illustrates an small abstracted KPN as a directed acyclic graph while equation 2.34 is the mathematical equivalent composite function. Since the mathematical formulation is quite cumbersome even for small networks, the visual DAG representation is preferred when illustrating most KPNs. The actual internal functionality of a process can still be better represented mathematically or as an algorithm.

$$\boldsymbol{o} = h\Big(\, g\big(\, f_1(\boldsymbol{i}_1), \, f_2(\boldsymbol{i}_2) \, \big), \; f_2(\boldsymbol{i}_2) \Big) \tag{2.34}$$

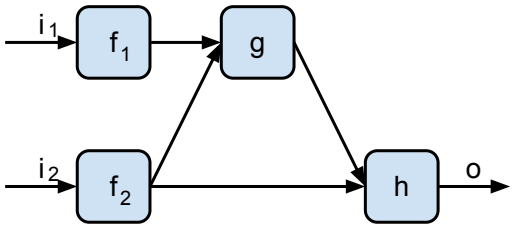


Figure 2.10: Functions f_1, f_2, g and h are applied to the input vectors $\boldsymbol{i}_1$ and $\boldsymbol{i}_2$, converted them into a single vector output $\boldsymbol{o}$.

2.3.2 Machine Learning and Data Mining Concepts

The following definitions within machine learning and data mining are based on Elkan's 2011 lecture notes on "Predictive analytics and data mining"[52].

Data mining is an interdisciplinary field that applies statistical methods and learning algorithms to real-world datasets. It is a goal oriented discipline with a focus more on creating applicable predictions to real-world problems rather than uncovering how the underlying system works. This makes data mining more of a predictive science rather than a descriptive one. Machine learning and data mining terminology will be used for the technical description of the system. The

following is a brief introduction to the concepts and terminology of data mining and machine learning.

Data Structure

Machine learning typically define a piece of data as a **datapoint**, instance, record or example. This paper will refer only to datapoints. Most often datapoints are fixed size vectors of real values. A datapoint indexed by i will give $\boldsymbol{x}_i = [x_{i1} \ \cdots \ x_{in}]^{\mathrm{T}} \in \mathbb{R}^n$. When you have a set of datapoints they are referred to as a **dataset**. By convention a dataset is often illustrated and stored as a matrix where each datapoint is a row, $X = [\boldsymbol{x}_1 \ \cdots \ \boldsymbol{x}_m]^{\mathrm{T}} \in \mathbb{R}^{m \times n}$. The individual values of the dataset X have the structure of (2.35).

$$X = \begin{bmatrix} \boldsymbol{x}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{x}_m^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} x_{11} & \cdots & x_{1n} \\ \vdots & \ddots & \vdots \\ x_{m1} & \cdots & x_{mn} \end{bmatrix} \tag{2.35}$$

A column in a dataset is referred to as a **feature**, the equivalent concept in a datapoint is called a feature value. This means that there are n features in a dataset. When referring to a feature in a dataset this refers to a common attribute that describes something about each of the datapoints. If a dataset describes a collection of cars, then two of the feature could be "top speed" and "mileage". The collection of features that describe the attributes of a dataset is called a **feature set**. In practice, a datapoint can be anything from a customer record to a sensor reading. A collection of time dependent readings is often referred to as a **time series**. This makes features and time series directly analogous. A vector containing readings from several time series at one point in time would thus constitute a datapoint. Time series predictions about the future is called **forecasting**.

Prediction using Supervised Learning

Prediction is in essence a process of trying to estimate some unknown feature associated with a datapoint. This feature is called a **label**. There can be many labels for one datapoint $\boldsymbol{x}_i$ such that $\boldsymbol{y}_i = [y_{i1} \ \cdots \ y_{ik}]^{\mathrm{T}} \in \mathbb{R}^k$. This gives a label structure for $Y \in \mathbb{R}^{m \times k}$ equivalent to the dataset X. This is illustrated more succinctly in 2.36.

$$Y = \begin{bmatrix} \boldsymbol{y}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{y}_m^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} y_{11} & \cdots & y_{1k} \\ \vdots & \ddots & \vdots \\ y_{m1} & \cdots & y_{mk} \end{bmatrix} \tag{2.36}$$

In practice the label tends to be the optimal action given the current state, but can also just be a prediction that needs some further analysis for decision making. Inferring the "true" labels of unlabeled datapoints based on a discretely labeled dataset is done by a **classifier**. However, when intentionally being ambiguous, this paper refers to both prediction models and classifiers under a collective term, **predictors**. For single datapoints, a predictor f can be defined as in (2.37), where $\boldsymbol{x}_i$ is defined as above, $\hat{\boldsymbol{y}}_i$ is a prediction of $\boldsymbol{y}_i$ and $\boldsymbol{\theta}$ is a set of parameters that defines the classifier. This notation extends to full datasets, see (2.38), where $\hat{Y} \in \mathbb{R}^{n \times k}$ is defined as in (2.36).

$$f(\boldsymbol{x}_i; \boldsymbol{\theta}) = \hat{\boldsymbol{y}}_i \tag{2.37}$$

$$f(X; \boldsymbol{\theta}) = \hat{Y} \tag{2.38}$$

A predictor takes a datapoint as input and outputs one or more labels. Prediction using multiple labels is called multi-labeled prediction or classification. This is accounted for in the label definition above. As indicated in (2.37) and (2.38), a predictor can be seen as a function that tries to map datapoints to labels. In those terms you can view the values which the input datapoints can take as the domain of a classifier. Conversely, the values that the labels may take can be viewed as the range of a classifier. In the case of discrete labels one use the term class rather than value, and talk about **classification** rather than prediction.

Predictors can employ learning algorithms based on numerous machine learning and statistical methods. When a predictor learns its parameters $\boldsymbol{\theta}$, or weights, one call this **training** the predictor on a dataset. In this project the emphasis is on **supervised learning** and a variation called **reinforcement learning**.

Supervised learning requires feedback about the performance of the predictor for it to be able to learn. Typical supervised learning indicates the error of a prediction. This can be used directly for learning. The training process indicates, in relative terms, how much and in what direction to improve the predictor's parameters. An example of supervised learning is stochastic gradient following on a linear regression problem.

Reinforcement learning, on the other hand, gives only utility as feedback. That is, only information about how good a solution is, not how to improve it. The utility can be any measure that indicates the performance of an action, or series of actions, in a given problem state. Evolutionary algorithms typically operate under these constraints.

Performance Evaluation and Overfitting

One critical issue with machine learning is the danger of **overfitting**. Overfitting happens when a predictor learns the inherent noise of the dataset. Overfitting is to find patterns in data that only occur from chance and calling it an coherence. Making a classifier overly complex, e.g., giving it too many parameters, can make it able to "fit" its parameters exceptionally well to the dataset it is learning from. Though the classifier can give up to perfect accuracy on this dataset, it is not able to generalize to new datapoints. This has repercussions with how one should evaluate a classifier's capabilities. There are many ways to limit or discovering overfitting, the ones relevant for this system is presented here:

1. Training and validation sets
2. Stop training
3. Reduce noise
4. Increase amount and diversity of data

Training and Validation Set. A common way to uncover overfitting is to split the labeled dataset into two separate datasets. One is called a **training set** the other a **test set**. The larger training set is used to train the classifier, while the smaller testing set is used to test it. It is important to hold the two datasets entirely separate and to use the test set only once in order to prevent **information leakage**. This can occur when a classifier gets information from the training set inadvertently. Having a test set is good practice for knowing when one is overfitting, however, how to avoid it is not as simple. Typically you would try to limit the complexity of the classifier such that it is not capable of learning noise. For some predictors it is possible to measure when overfitting starts while learning. This is done with a third **validation set** which the predictor tests on while training. When the results on the validation set starts to deteriorate training is terminated.

ANN Training. ANNs naturally overfit data if they are run too long on the same test data. The ideal is to discover the general patterns that is common to all of the data, not to fit perfectly to just this set of training data. To avoid this overfitting the training must be stopped at the right time. This ideal stopping point is the instant when the error on the training set still decreases while the ANN does no better or worse on a general validation set. Thus to find this point the ANN will run against a validation set in addition to the training set. The ANN will however not be adjusted after the validation set as this would contaminate it. When the performance only increase on the training set and not the validation set the training will be stopped, thus eliminating this source of overfitting[140], see

figure 2.11 for an illustration.

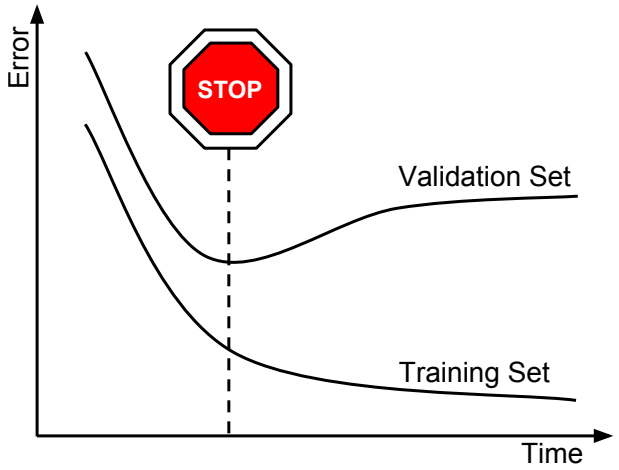


Figure 2.11: Here we see a diagram of the error from the validation set (red) versus the error from the training. At the point marked the training should be stopped to avoid overfitting.

Evolutionary Overfitting. The evolutionary training process needs also a stopping point for training. This will be set to an error target, but with an additional limit on the total number of generations for performance reasons. Another approach to discover overfitting in evolution is to measure the homogeneity in the population, if this becomes too homogeneous it indicates that the process is overfitting.

Noise Reduction. The data fed into the model will have a certain amount of noise. An important issue is to predict the underlying trend and not the noise of the data. One approach to avoid this noise and thus limit overfitting is to remove it a priori. There are several approaches to removing noise, from simple moving averages to neural networks. Several such methods will be employed and all or an aggregation of these will be presented to the portfolio module. In this way the choice of which noise method is the best is transferred to the system as the input selection is under evolutionary pressure. A potential problem here is that all "noise" in the price could potentially be exploited by the system, depending on what resolution and scope one have the definition of what is noise will vary. Thus this definition of noise will have to be adapted to the trading resolution of the system.

Amount of Datapoints. As discussed earlier, most of the system is evolvable. This does not mean all will be evolved at all time. For some of these parameters

an adequate value will be found and then fixed. Other parts will only be run at certain intervals of data. The more parameters in an evolutionary search, the larger are the degree of overfitting. This is the main reason to avoid evolving to much of the system at any given time.

Diverse Data. The other mentioned aspect is to use diverse data. Even though have many similar inputs, e.g. different noise reduced price series etc., it still has several fundamentally different categories of data. An issue here could arise if the evolution of inputs ends up with a relatively homogeneous set of inputs. This is easy to discover and can then be prevented by a fitness reward for having a diverse selection of inputs.

2.3.3 Regression methods

This section will present different regression models planned to be used in the system. Details of their part in the system will follow in the results part. To motivate the following methods we will first look into linear regression, then move on to the autoregressive methods ARCH and GARCH. Note that the following models will be presented in a data mining context, thus omitting noise variables. This is because all the models are in terms of the actually estimated values and not true value equals to noise added to an estimate. Though, the noise in data mining would be equivalent to error of prediction.

Linear Regression

Linear regression tries to learn a vector of weights $\boldsymbol{\theta}$ that are linearly combined with datapoint $\boldsymbol{x}_i \in X$. As previously described, a classifier can output multiple, specifically k, labels $y_{ij} \in \boldsymbol{y}_i \in Y$ for some datapoint $\boldsymbol{x}_i$. However, the classical linear regression classifier outputs only one single real valued label, such that $\boldsymbol{y}_i = y_i \in Y = \boldsymbol{y} = [y_1 \ \cdots \ y_m]^{\mathrm{T}}$. A linear regression classifier can thus be defined as (2.39). An intercept, or constant term, is often used in linear regression. Without an intercept a linear regression model is forced to intersect the origin. Adding an intercept while making the mathematics compact can be done by augmenting the training set with a feature of only ones, i.e., the one vector $\mathbf{1} \in \mathbb{R}^m$, such that $X' = [\mathbf{1} \ X] \in \mathbb{R}^{n \times m+1}$. This operation is called a **one-augmentation** of the dataset X. Performing this operation requires that the number of weights in $\boldsymbol{\theta}$ be increased by one, yielding $\boldsymbol{\theta'} \in \mathbb{R}^{m+1}$. The classifier remains identical to that of (2.39), i.e., it can be used directly, $f(X'; \boldsymbol{\theta'})$.

$$f(X; \boldsymbol{\theta}) = X\boldsymbol{\theta} = \boldsymbol{y} \tag{2.39}$$

Actual learning of $\boldsymbol{\theta}$ can be done with (stochastic) gradient following using a differentiable error such as squared error. That is, an error function can be used

that utilizes the differentiability of the Euclidean distance between prediction and labeling as in (2.40).

$$\varepsilon = ||y - f(X; \boldsymbol{\theta})||_2^2 \tag{2.40}$$

Linear regression per se is not used directly in any system. It is however an important basis for many of the price prediction techniques used.

Non-linear Regression, using the Kernal Trick

The problem with linear models, like linear regression, is that there might be non-linear relations that cannot be represented linearly. Trying to fit a straight line to dataset of mortality rates versus blood pressure might yield horrible results. This is because neither very high nor very low pressure is good. It would be much more reasonable to fit a polynomial function to this problem. This leads to looking for ways to transform a non-linear problem to a linear one. That is, one can translate a linear regression model into a **non-linear regression** model by transformation of the feature space it is applied to. The following procedure is a general concept in machine learning called the **kernel trick**. In essence you push the dataset into a higher dimensional non-linear space in the hope of capturing linear relations.

An example linear model could be described by $\boldsymbol{x} = [1\ x_1\ x_2]^{\mathrm{T}}$ and $\boldsymbol{\theta} = [\theta_0\ \theta_1\ \theta_2]^{\mathrm{T}}$ with classifier $f(\boldsymbol{x}; \boldsymbol{\theta}) = \boldsymbol{x}^{\mathrm{T}}\boldsymbol{\theta}$. If one transform $\boldsymbol{x}$ to some higher dimensional polynomial variant $\boldsymbol{x}' = [1\ \ x_1\ \ x_2\ \ x_1^2\ \ x_2^2\ \ x_1x_2]^{\mathrm{T}}$ and add more parameters such that $\boldsymbol{\theta}' = [\theta_0\ \cdots\ \theta_5]^{\mathrm{T}}$, then the classifier $f(\boldsymbol{x}'; \boldsymbol{\theta}')$ is still linear, but we're doing a non-linear classification task. The specific transformation does not need to be polynomial as in this example; any transformation can be used.

Rationale for Multivariate Non-linear Models

Forecasting models varies along two dimensions. They can be linear or non-linear as described above and they can be univariate or multivariate. That is they can predict based on one variable or several. Several studies claim non-linear models is superior to linear in financial forecasting [74; 105; 139]. Most techniques described in the chapter and used in these systems does non-linear modeling, though a few linear is described and used in combination. Much of the rationale behind the approach this paper takes is to combine data sources in an attempt to identify and exploit patterns. This approach of combining data require multivariate components. Some univariate are used, but only as input to multivariate ones. Zhang, Cao and Schniederjans found that Neural Networks, a common non-linear model that also is a central technique in this approach, outperformed traditional linear techniques both as univariate and multivariate [158]. The multivariate model, which got fundamental signals in addition, outperformed the univariate.

Autoregression

The purpose of using regression in forecasting is to predict the next value of a time series. **Autoregressive models** work much in the same fashion as linear regression. The difference lies mainly in how you define the input data. Normally the internal structure and correlation of a datapoint's feature set is not of importance, this is in contrasts to autoregressive models that have feature sets defined by a window that slides along a time series. A datapoint, i.e., a row of the final dataset, is simply a snapshot of all the values within a window of the time series. For each snapshot the produced datapoint is stored and the upper and lower bound of the window incremented, finally the process is repeated. If one let a time series be defined by a vector $\boldsymbol{z} = [z_1 \ \cdots \ z_t \ \cdots \ z_T]^{\mathrm{T}}$ and use a window with size p, then (2.41) describes the transformation process needed to convert $\boldsymbol{z}$ into a dataset $x_{ij} \in X$ applicable to the previously described linear regression classifier. Thus, with one-augmentation of the dataset, the autoregressive model reduces to a linear regression task with a classifier defined by (2.42).

$$\begin{aligned} x_{ij} &= z_t \\ t &= i + j - 1 \\ i &\in \{1, \cdots, T - p + 1\} \\ j &\in \{1, \cdots, p\} \end{aligned} \tag{2.41}$$

$$\begin{aligned} \hat{z}_{t+1} &= f(\boldsymbol{x}_i; \boldsymbol{\theta}) = \boldsymbol{x}_i^{\mathrm{T}} \boldsymbol{\theta} \\ i &= t - p + 1 \end{aligned} \tag{2.42}$$

It is possible to forecast further into the future by creating new datapoints as

Table 2.7: A more visual illustration of the sliding window time series conversion to a dataset.

$z_t = x_{ij} \in X$	$j = 1$	$\cdots$	$j = p$
$i = 1$	z_1	$\cdots$	z_p
$i = 2$	z_2	$\cdots$	z_{p+1}
$\vdots$	$\vdots$	$\ddots$	$\vdots$
$i = T - p + 1$	z_{T-p+1}	$\cdots$	z_T

prediction is performed. That is, when a prediction is made for one time step into the future, $T + 1$, the value of this prediction can be used to create another datapoint $\hat{\boldsymbol{z}}_{T+1} = \hat{\boldsymbol{x}}_{T-p+2}$ with the structure described in (2.41). Now applying this result to the time series predictor can yield another time series value, now at

time $T+2$. This can be carried continued as far as needed. Of course, the further you go into the future the less reliable the predictions are.

Training of the autoregression weights can be done in numerous ways. One common way is using Yule-Walker equations to find lag coefficients and solving directly by inversion. Another way is to solve the linear regression problem by maximum likelihood using stochastic[6] gradient training [53].

One trading system is based on autoregression. This uses autoregression to predict price movements based on technical data and historical prices.

Moving Averages

A simple filtering mechanism for time series, sometimes used for rudimentary forecasting, is the **moving averages** method. The setup for moving averages works much in the same way as the autoregressive model just developed in (2.42). The difference being that there is no learning. The weights are fixed a priori, or determined by the development of the time series. The simplest form is the rolling mean. It simply finds the mean value of a sliding window and outputs this filter value as a prediction. This can be modeled as $\theta_j = 1/q$ where q is the window size.

Moving averages are abundant in the system. There exists trading system that is based solely on moving averages like MACD. It is also used in any case where data is needed in a historical context; the moving average gives the possibility of balancing reactivity to new data with the information in old by setting the length of the window.

2.3.4 SVM

Support vector machines (SVM) are a set of supervised learning methods used for classification and regression analysis. SVMs are linear non-probabilistic binary classifiers, i.e., they can classify datapoints as one of two classes. This is done by intersecting a hyperplane through the feature space that separates one cluster of similarly labeled training data from another. The SVM learns the parameters for this hyperplane by maximizing the margin from the hyperplane to the two training data clusters.

More advanced SVMs will use soft margins[47] that react gracefully to abnormally labeled datapoints and semi-overlapping data clusters that cannot be separated by a simple hyperplane. Though, the more overlap between the two clusters of datapoints, the worse a SVM will do. This can be mitigated by transforming the

[6]The term "stochastic" in stochastic gradient descent refers to how the learning method is performed. It is a stochastic approximation to gradient descent, where updates are performed based one datapoint at a time.

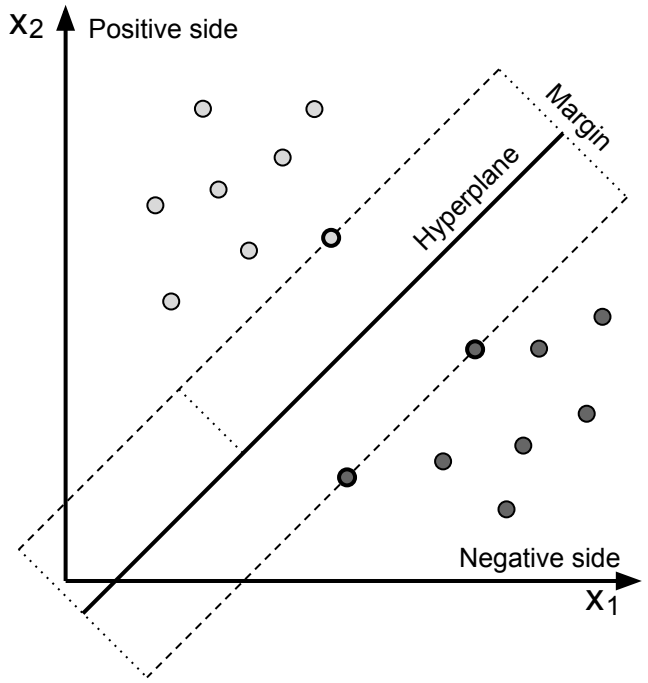


Figure 2.12: Illustration of a margin maximizing SVM hyperplane, in a two-dimensional feature space, separating positive- and negatively labeled datapoints.

initial feature space into a higher dimensional feature space by using the kernel trick, as described with the non-linear regression example. However, this increases the danger of overfitting, as with most models that increase the number of parameters. For more details on the technical aspects of the SVM consult section 7.2 in the appendix.

Support Vector Machines (SVM) are very popular in modern systems for prediction problems. Several studies show SVMs outperforming ANNs and other common techniques at this task [115; 137]. It should be noted that the ANNs most studies test against are standard feed-forward networks with backpropagation. A study was done comparing RNN, SVM and ARMA by Thissen et al [142]. They found that SVM and Elman were largely equal in performance on time series predictions and outperformed ARMA. SVM was also found to require much less training data than the Elman network. Due to these satisfactory results an SVM is applied to the prediction module as an agent.

Support Vector Regression (SVR) is the regression variant of SVM using the

same principles, but returning the line instead of only the classification. Thus it can return the actual degree of a class a datapoint is found to be.

Both SVM and SVR are used as time series predictors in several trading system. These are among the most prominent and promising time series predictors in use and have as such been used more extensively than the older techniques.

2.3.5 Evolutionary Algorithms

An **evolutionary algorithm** [63] is a metaheuristic optimization algorithm that draws inspiration from biological evolution. Biological evolution works through natural selection, where individuals in a population of a species compete for the possibility to reproduce. Individuals who are able to create many viable offspring are said to be fit. This means that their inheritable traits, i.e., phenotypes, will tend to be more pervasive than their less fortunate counterparts in the following generation. Over many generations this will tend to allow a population to adapt to changes in the environment, and increase its viability. Notice that changes in phenotypes does not occur in individuals, but rather between generations. This biological explanation of evolution is analogous to evolutionary algorithms used for parameter optimization.

In evolutionary algorithms an individual can be viewed as a solution method to a problem. The problem must be repeatable and give feedback about the utility of the solution method. In evolutionary terms the utility that an individual generates is called fitness. The fitness that an individual can produce is determined by its parameters, i.e., phenotype.

There are many specific ways to perform evolution; the following is a typical example. A population of randomly generated individuals is created. The individuals, i.e. solution methods, are applied to the problem at hand and fitness levels are recorded. First, a subset of the worst performing individuals is removed from the population. Then, a subset of individuals with phenotypes that yield the highest fitness is allowed to reproduce. The reproducing individuals are naturally called the parents of that generation. Reproduction is performed by creating multiple clones of the parents. The parameters of the clones are given random, yet modest, adjustments. This action makes them children of their parents, rather than clones. The children are added to a fresh population and another round, i.e., generation, is performed. This continues until some stopping criteria are reached. An illustration of this algorithm in practice is shown in figure 2.13.

There are many important variations of this basic evolutionary algorithm. One of the more important variations is the use of a genotype as the medium of mutation. A genotype is a discretely encoded version of a phenotype. A typical type of code

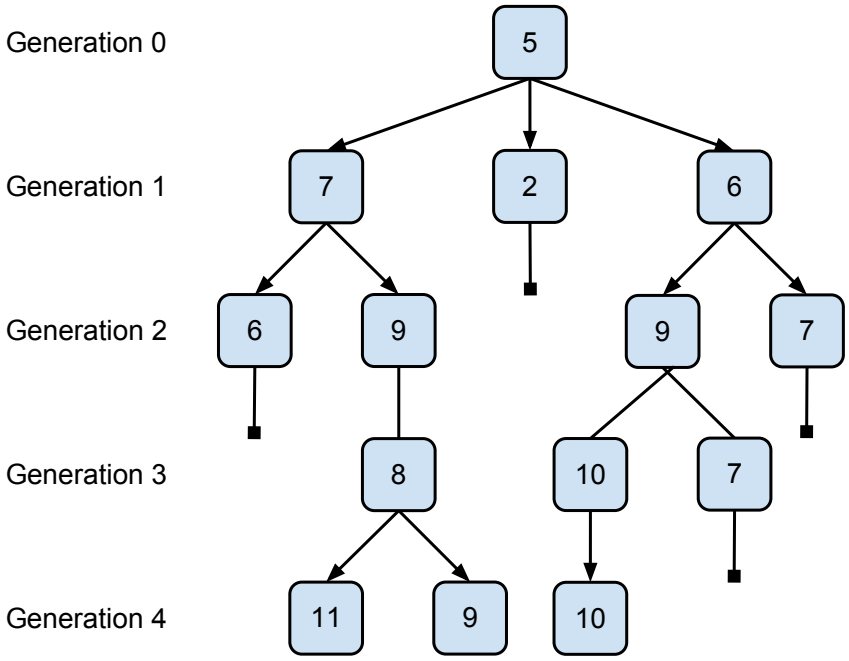


Figure 2.13: An illustration of four generations of an evolutionary algorithm where the two most fit individuals are allowed to reproduce while the rest are terminated. The phenotypes are randomly mutated with a uniform distribution over $[-3, 3]$.

could be an array of binary values. In biology, DNA is the genetic code describing a genotype. The benefit of using a genotype is that it tends to increase a populations ability to find better solutions. This is because small changes in a genotype might give a drastically different phenotype, thus nudging the solution search out of local minima.

Other important variations include using more severe mutations, like cross-over, and changing the termination or breeding protocol.

Evolutionary algorithms are good at performing many parameter adjustment tasks, like finding suitable parameters of a classifier. Though, it is even more suitable for meta-parameter optimization, i.e., optimization of the structure of a classifier rather than its lower level parameters. It is common to let evolutionary algorithms take responsibility of adjusting the proverbial "shape" of a classifier and use some other learning method to find the lower level parameters.

Evolutionary algorithms are used as a meta-parameter optimizer in the framework. Some experiments with evolution as semi-online learning is also done, though this

is not yet implemented as a formal trading system and will be followed further in future work.

2.3.6 Artificial neural networks

Artificial neural networks (ANN) are computational structures inspired by the neural networks of biological brains. The original perceptron model was developed by Rosenblatt in 1958 [122]. This was picked up in 1986 when David Rumelhart, Geoffrey Hinton and Ronald Williams published "Learning Internal Representations by Error Propagation" [123]. They purposed the multilayer neural network with non-linear, but differentiable transfer functions and reasonably effective training algorithms. When writing ANN in this paper this refers the general technique and includes all the different types discussed and used in this project.

Artificial neural networks consist of a set of interconnected neurons that interact with each other by sending variable strength activation signals. Each connection has an associated weight, i.e. parameter, which modulates the activation signal. The activation signals from all connections to a neuron are combined by a transfer function. This function is typically just the summation of the individual weighted activation signals. The transfer function's output is passed to an activation function which fires if the output surpasses a threshold. Some implementations use an S-shaped activation function, such as a sigmoid function, instead of binary activation. The resulting activation function output is then passed along the neurons outgoing connections.

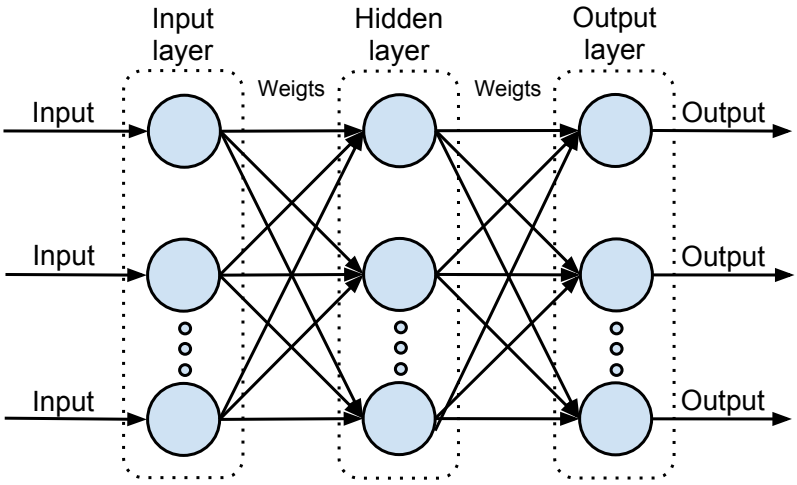


Figure 2.14: A feed forward neural network with a single hidden layer.

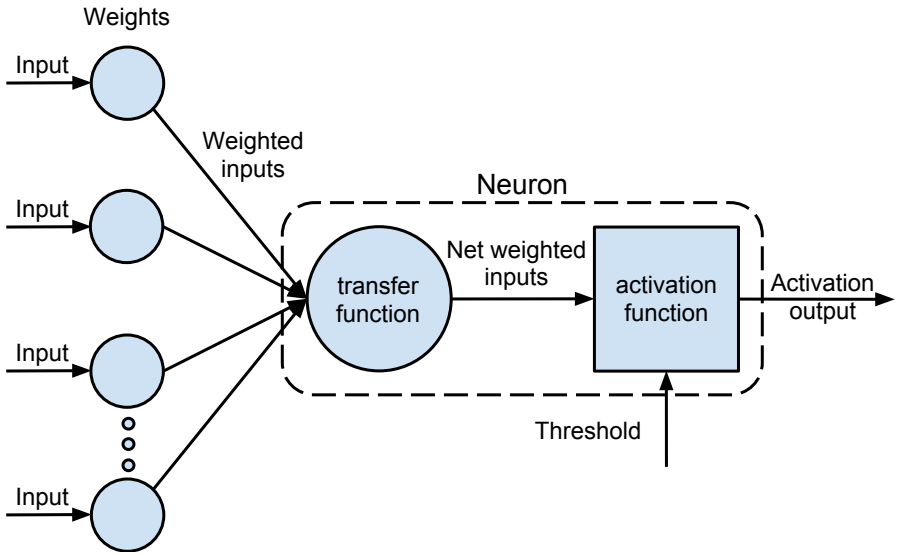


Figure 2.15: A neuron sums multiple weighted inputs from other neurons and applies it to an activation function. If the net weighted input exceeds the activation threshold, then the activation function will fire a signal known as the activation output.

ANN Types

Here the different ANNs relevant for this paper is presented. This is a small subset of the different ANNs that exist, but this subset has been chosen for their relevance in this domain. There are four ANN types at use in this project; they again fall into two categories. There aare the common traditional multilayer perceptron neural networks (MLPNN) of which feedforward neural network and recurrent neural network are members. Then there is the more advanced radial basis function networks (RBF) of which probabilistic neural network (PNN) and general regression neural network (GRNN) is used in this project. For an overview of these and their relationship see figure 2.16

Feedforward Networks. Feedforward neural networks (FNN) are thought of as the standard ANNs. They are connected like a directed acyclic graph, with an additional restriction. All neurons are parts of one layer, and all neurons in one layer can only connect to one other layer, creating a chain of layers. For an example with one hidden layer, see figure 2.14.

Recurrent Neural Networks. Recurrent neural networks (RNN) are similar to multilayer feed forward networks with the addition of cycles between layers. There are several types depending on where the cycles are, but the common purpose is

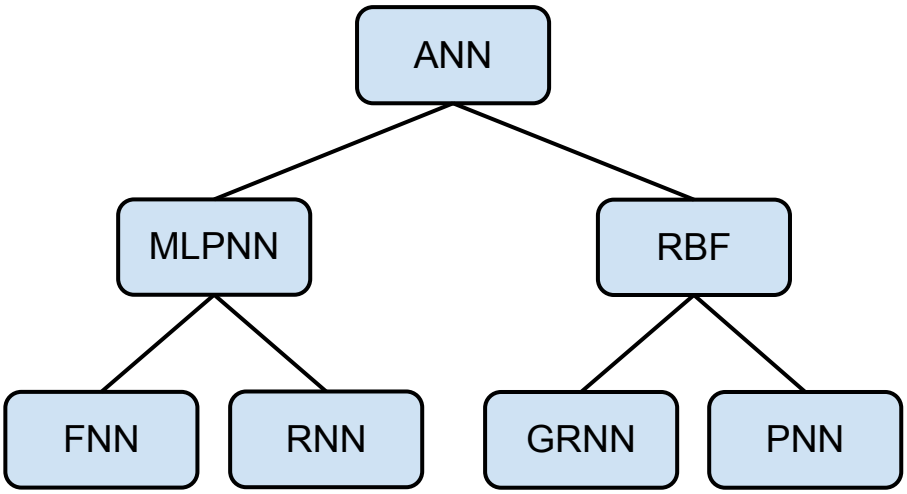


Figure 2.16: The hirearki of the ANNs used in this system

to provide memory and attractor dynamics in the network [150]. One of the drawbacks with recurrent networks is that they tend to have complex and often obscure behavior and are considered harder to train and understand. Jordan[88] was the first to create such networks, but Elman's [54] adaptation is more commonly used. The main difference between feed-forward networks and RNNs is that the latter naturally takes into account previous signals and can thus achieve memory and a "sense of time". This makes RNNs well suited to discover temporal patterns [21; 71; 72; 86]. In an Elman network the first hidden layer feeds its input to a context layer as well as the next layer. This context is then fed back to the input as part of the next datapoint, see figure 2.17.

RBF Networks. Both general regression neural networks (GRNN) and probabilistic neural network are branches of radial basis function (RBF) networks[109]. RBF network have been shown to have a higher computation precision as well as faster convergence rate than back propagation network [110]. The GRNN and PNN model was proposed by Specht, and is capable of estimating any function of historical data [136]. These two basically have the same architecture. Probabilistic networks perform classification where the target variable is categorical, whereas general regression neural networks perform regression where the target variable is continuous. The GRNN has four layers 2.18. General regression neural networks (GRNN) are used in some papers for time series prediction with good results [98; 152]. PNN has also shown good results on time series prediction [124].

In the **input layer** as with normal ANNs each neuron corresponds to each predictor variable. The **hidden layer** has one neuron for each case in the training

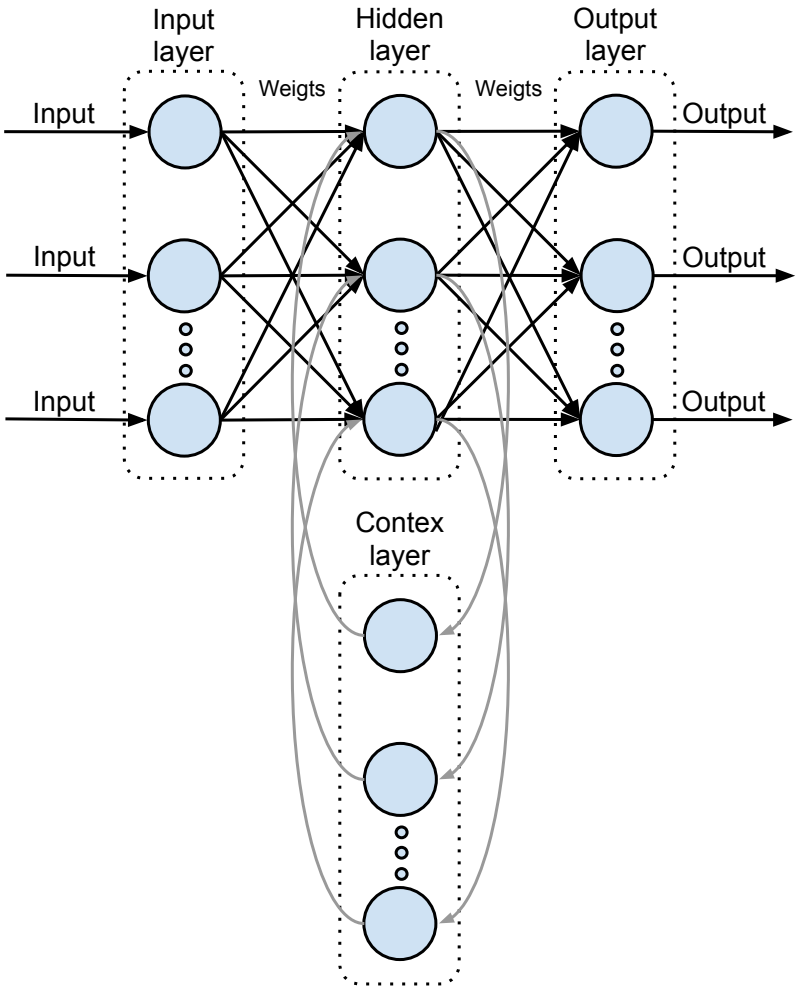


Figure 2.17: The architecture of an Elman recurrent neural network

dataset and uses an RBF function on the Euclidean distance between the current case and the neuron's center. For GRNNs the next layer, the **summation layer** only has two neurons. The denominator neuron sums adds up all weight values from the hidden layer neurons. The numerator neuron adds the weight values multiplied with the actual target value for each hidden neuron. In the last layer the lone **decision neuron** then divides the numerator by the denominator to find the predicted target value.

All the ANN types mentioned here are in use as time series predictors in trading systems. The MLPNNs are also used as portfolio selectors by mapping the portfolio positions to the output neurons.

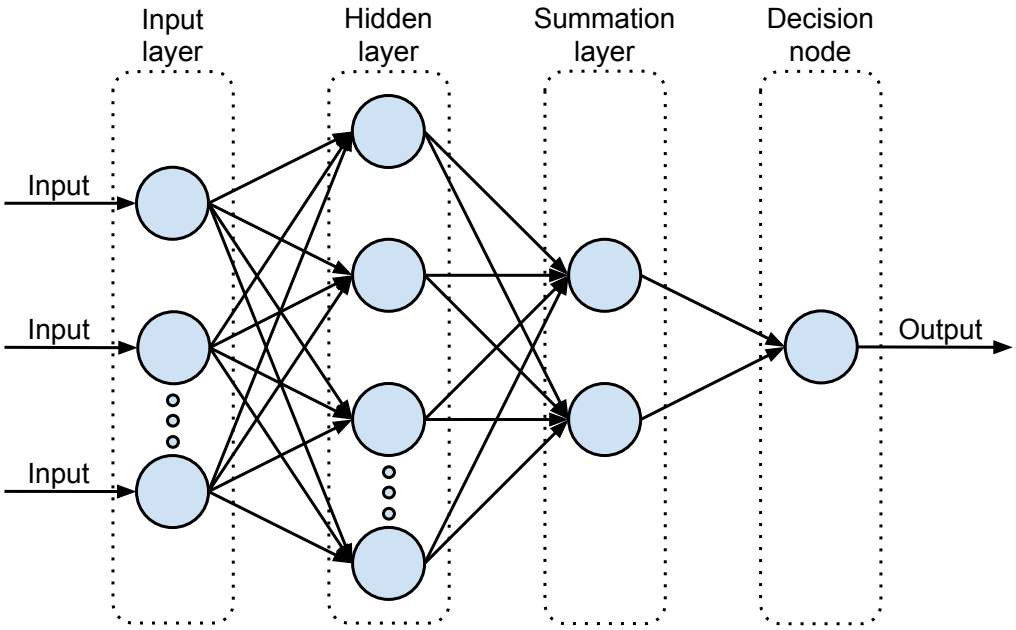


Figure 2.18: The architecture of a General Regression Neural Network

Training the Neural Network

The methods used to train the system are of the same level of importance as the choice of predictors themselves. Both prediction and portfolio agents need training to be effective. Due to their differing structures some need different learning methods. However, some learning methods like evolution is universally applicable to meta-parameters, and may even be used for initial application on learning parameters. For low level parameter optimization in neural networks variants of backpropagation can be used. For parameter search, few techniques can compete with the ease and effectiveness of evolution. Evolution perform well at broad search, especially where the search space are complex. Evolution is, however, weak at fine tuning. This is where backpropagation shines, thus making the two techniques ideal for combination into a hybrid solution. Many studies have shown that this hybrid approach performs well [29; 90; 144; 149; 157]. It is both well tested and flexible, and thus worthy approach to apply. There are, however, others with several advanced techniques that should be considered. Alternatives to the initial technique will be considered if the hybrid model does not deliver.

Training a neural network involves adjusting its weights (and thresholds). This can be done with regard to some metric of performance. Typically this is done by minimization of the error between a target output, i.e., label, and the predicted

output. In practice this is usually done with a specialized gradient decent algorithm such as **backpropogation**. Since gradient decent is a greedy algorithm, backpropogation suffers from the same common problems associated with this technique. It may get stuck in a local optima of multimodal solution spaces and it need a differentiable error function[153].

Aside from backpropagation there are conjugate decent algorithms, also based on gradient decent, with much the same advantages and drawbacks. To overcome the disadvantages with the gradient decent approach evolution can instead be used on the connection weights. Evolution can effectively find near optimal set of weights without computing gradient information. A disadvantage with evolution is that it cannot guarantee a optimal solution and often have problems when it comes down to fine-tuned search. Thus a hybrid solution between evolution and a gradient decent based approach is common, and shows promising results[153]. Evolution does the rough search and then may use gradient decent for the last part of the search.

Another problem for the classical backpropagation is the degree to which weights are adjusted. **Resilient propagation** is a modified version of backpropagation that deals with this issue. Contrary to gradient decent, resilient propagation only uses the sign of the change, not the magnitude. Resilient propagation keeps individual delta values for each weight and uses these when the weights are to be adjusted. The gradient is then used to determine how much the deltas should be adjusted [84; 120].

ANNs are very prone to overfitting as they will continue to adjust until they perfectly fit any input data [155]. To avoid this problem it is important to stop the training when at the point where the generalization ability starts to fall. It is also important to have a large set of data relative to the size of the network.

Evolving ANN Architecture

When referring to an ANN's architecture this include the topology and the transfer functions of a network. This includes of how many layers are used, how many nodes in each layer has and how the neurons are interconnected. The architecture of a network is crucial for its information processing capabilities [153]. Traditionally the architecture design has been a human expert's job.

When presented with a large solution space and no known algorithm to easily create an optimal architecture it is common to use trial and error. There has been done some research toward automatic design by constructive or destructive algorithms [135]. Angeline et al. [15] points out the weakness of such approaches as they tend to end up at local optima, they also only investigate a restricted topolog-

ical subset rather than the complete architectural search space [153]. More recent research on automatic architecture generation focuses more on genetic algorithms. These are less computationally intense, cover a large search space and are less prone to over-design than those designed by human experts[34]. The search space to find a suitable ANN architecture is infinitely large, non-differentiable, complex, deceptive and multimodal[153]. All these characteristics make it hard or impossible to use traditional algorithms like gradient decent. This leaves evolutionary algorithms using genetics a better chance to find good solutions to the architecture problem.

Miller et al. shows that evolution is well suited for finding neural network architectures. This is because evolution is adept at this type of search spaces[107].

- The search surface is infinitely large as one can always add more neurons
- The search surface is nondifferentiable as there are a discrete set of neurons
- The search surface is noisy since the mapping between architecture and performance is indirect.
- The search is deceptive as small changes in architecture can make large differences in performance
- The search surface is multimodal since different architectures may have similar performance

Yao shows in his review of evolutionary ANNs (EANN) that evolution has excellent performance on finding good architectures for ANNs. Furthermore he recommends evolving the weights in the same process[153]. Backpropagation is the quintessential training method for ANNs and a combination of these two approaches in hybrid form has, as mentioned, been successfully validated in many studies[29; 90; 144; 149; 157].

2.4 Related Work

2.4.1 Preliminary Project

This system is a continuation of an earlier design project. In this project the preliminary design and goals for this system was constructed [14]. In short the result from this project was the fundamental design and architecture used in this project as well as which techniques to focus on. The techniques found were artificial neural networks, more specifically ANN, RNN and PNN/GRNN and support vector machines. Evolution was decided to be a central learning method along with the native methods for each technique. It was decided on a highly modular design to

facilitate for testing of many configurations. The actual final implementation will be presented in chapter 3, where important changes and why they were made will be discussed.

2.4.2 High Frequency Trading

High Frequency Trading (HFT) as mentioned earlier are only similar to this approach in the aspect that they are algorithmic training systems. Aside from this they usually employ very naive investment strategies in order to capture small margins and then repeat this at a very high frequency. What we can learn from these systems is that complexity is not necessary to generate large returns on the market. These systems rely on highly efficient algorithms as well as expensive hardware. This is an increasingly competitive domain with shrinking margins. Most actors are supported by large powerful organizations with the capital required to operate in this area. As these conditions make this an unattractive market for this project, HFT systems will not be discussed any further.

2.4.3 Time Series Prediction

Many systems aim to predict the price or some other feature of the stock market. A large difference between these systems is whether they take fundamental data into account. Many predict only based on previous price data, and thus are purely technical. Others add some fundamental data in order to improve the prediction. The latter is however less common. A popular technique for prediction seems to be different regression methods. The traditional one is Autoregressive Moving Averages (ARMA), these are however seldom used as other than benchmarking in modern papers. The most common financial time series predictors use support vector machines (SVM), artificial neural networks (ANN), genetic programming (GP) or other statistical pattern recognition, often combined with fuzzy logic.

Artificial Neural Networks

ANNs are central techniques in this system and will thus be discussed at length. Studies suggesting techniques that can supplement ANNs will also be discussed. Frank et al. gives an introduction to time series prediction with neural networks, with important issues like window size and sampling, they conclude that the window size is a crucial determinant of the quality of the prediction [64]. Adya and Collopy finds in their review of neural networks applied to forecasting and prediction that out of 22 approved papers, 19 where positive to neural networks, while 3 where negative[11]. Zang et al. also agree that ANNs gives satisfactory performance in forecasting [156]. Yao and Tan concludes that ANNs are suitable for forecasting of financial time series such as stock indices and foreign exchange rates

[87]. Chan et al. use a neural network with conjugate gradient learning and multiple linear regression (MLR) weight initialization to do financial forecasting[100]. They find it is possible to model stock prices using a three-layer ANN and that MLR speeds up the efficiency of BP. Although many studies uses standard feedforward networks, it is clear from other studies that specialized ANNs generally outperform these, though often at the cost of added complexity.

According to Saad et al. there are several specific classes of ANNs that performs very good at time series predictions. They did a comparative study of three ANN types specifically apt at time series predictions[124]. These are Recurrent Neural Networks (RNN), Probabilistic Neural Networks (PNN) and Time-Delayed Neural Networks (TDNN). They conclude that RNN are slightly superior, but harder to implement. They use Extended Kalman Filters to train the RNN. Other studies agree that RNNs are well suited for time series predictions [21; 71; 86]. Leung compares GRNN against the conventional multilayered feedforward neural network and finds that GRNN perform better than the other ANN as well as a set of econometric techniques at predicting foreign exchange rates[110].

Evolutionary Algorithms

Evolutionary techniques like genetic algorithms (GA) or genetic programming (GP) are often used for financial time series predictions. These are usually used in combination with other techniques, typically ANNs. Hassan, Nath and Kirley creates a fusion model using GA[78]. The ANN is used to generate inputs to the HMM and GA is used to optimize the HMM parameters. They find it to perform equal to the ARIMA model. Kim and Han uses a hybrid model ANN and GA where GA is used to evolve the weights in the network [94]. They conclude that evolution of other parameters parallel with the connection weight worked well, but that their fixed architecture is a limitation. Kim and Shin investigates different hybrids of GA and ANN for their effectiveness of discovering temporal patterns in stock markets[93]. They use adaptive time delay neural networks (ATNN) and time delayed neural networks (TDNN) with GA. They find that the accuracy of the integrated approach is better than standard ATNN and TDNN as well as RNNs. Armano et al. proposes a novel hybrid ANN and GA architecture for stock index forecasting [17]. They create a set of experts that do prediction or classification; these use both GA and ANN techniques. They test it on two stock markets and receive good results.

Support Vector Machines

Support vector machines are starting to dominate the field of time series prediction. Most new systems test these with ANNs as benchmarks, and they generally perform better than standard ANNs. Sapankevych and Sankar [126] do a review of

time series prediction using SVM. They point toward many studies showing that SVMs perform better than different ANNs at this task. Several hybrid approaches has also been proposed. Bao et al. propose a self-organizing map (SOM) combined with SVM [24] and find that the hybrid outperform pure SVM. Pai and Hong propose a Recurrent SVM with Genetic Algorithms for parameter search[115]. In their design the SVM act as a layer in their Jordan RNN see figure 2.19. A similar approach is used by Vikramjit et al. [108] where they integrate a RNN and a least squares SVM and achieve good results on text classification. An even more complex model is proposed by Shi and Han [133]. They have a RNN system with a Support Vector Echo-state machine. They receive satisfying results on benchmarks and some real data, but it is not tested on financial data series. Support Vector machines are a relatively novel machine learning approach that shows great promise, and most applications of this new technique is in financial forecasting [126]. Yang and Zhang [152] compares a Support Vector Regression (SVR), a least squares SVM (LS-SVM), BPNN, RBF network and a GRNN for predicting vibration time series. For short time prediction they find that the SVR outperform all, on long term prediction however, the RBF network perform best.

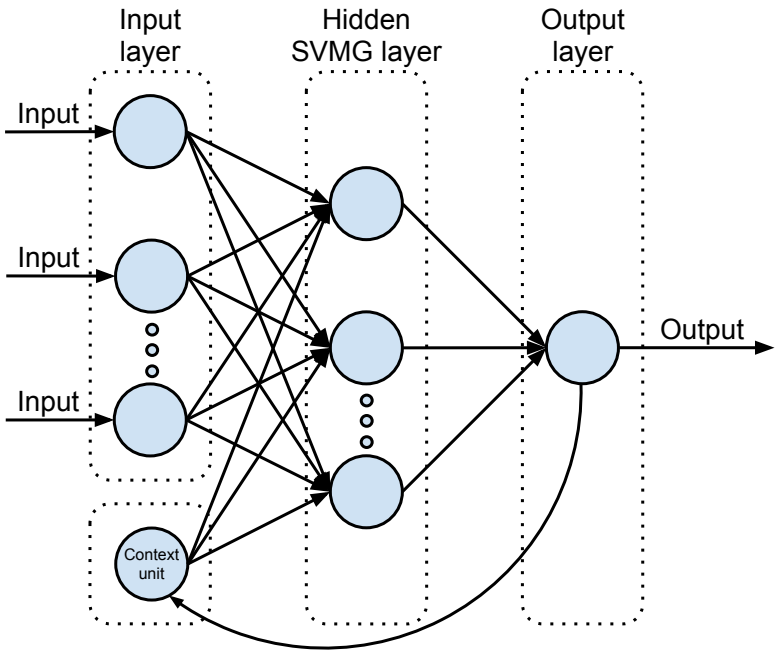


Figure 2.19: This figure shows Pai and Hongs Recurrent SVM with Genetic Algorithms (RSVMG). We can see that their SVM is placed inside a Jordan recurrent network.

Statistical Models

There have been done some promising research into prediction with non-linear statistical models. Li, Liu and Le combines General Regression neural networks (GRNN) with GARCH models and shows it perform better than any of the two alone [98]. Santini and Tettamanzi find that pure genetic programming give very good results, and actually claim that combining this with GARCH, Markov processes and probabilistic learning degrades their performance[125].

Zhang use a hybrid ARIMA and neural network model to forecast time series and finds that this hybrid outperforms any of the models alone[154]. ARMA or ARIMA is used in many other articles, but usually as a benchmark.

2.4.4 Portfolio Optimization Systems

The time series prediction section is concerned with only predicting of single assets or time series. The system is designed to select portfolios; in this section relevant systems for this use will be presented. Only one system was found using neural networks for actual portfolio selection in the same way as this system plan. This is a patent from 1998, filed by Dean S. Barr [26]. He describes a system somewhat similar to this system. He has three layers where one is a layer of ANNs that take in pre-processed market data, see figure 2.20. He has his own portfolio construction module, so it is not clear whether the ANNs actually output portfolios like this system will do.

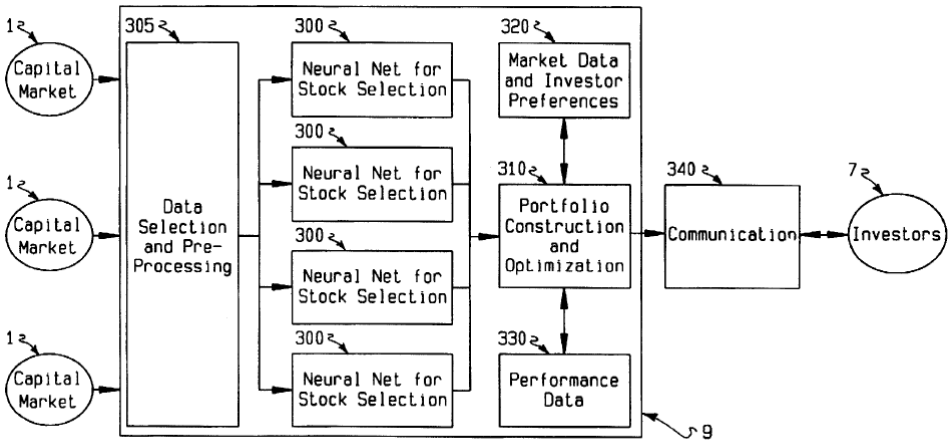


Figure 2.20: This the architecture of Barr's Portfolio optimization using ANNs

Zhang and Hua use neural network and Markow decision processes to achieve bankruptcy control and dynamic portfolio selection according to Markowitz's mean-

variance portfolio selection model[159]. Their results show their system to outperform both pure GA and Linear Programming. Lin and Liu show in their paper that portfolio optimization based on the Markowitz model with minimum transaction lots is NP-Complete, but that a good solution can be achieved using evolutionary heuristics [99]. Zimmerman and Neueneier use neural networks which adapts the Black/Litterman portfolio optimization algorithm. The funds are allocated across securities while simultaneously complying with a set of constraints posted by the user. Gaivoronski, Van Der Wijst and Krylov proposes techniques for optimal selection of portfolios based on a perception of risk and target measures [70]. Their paper further discuss how to optimally rebalance a portfolio given both fixed and proportional transaction cost through a mixed integer LP problem.

2.4.5 Criticism of Sources

Most of the papers found are purely academic. Systems are mostly tested against known benchmarks or some sort of market simulator. None have been found that claims to actually generate a profit from their system. This issue is probably due to the fact that inventors of successful novel approaches would lose much of their advantage by sharing their ideas. Thus they have little or no incentive to publish a practical and working system. There exist many systems out there claiming to employ both ANNs and evolution in their proprietary systems. This is however a field riddled with fraud and it does not seem to be a legitimate claim.

There also seems to be quite a large bias in the literature. First, very few admit failure in their papers. This might be due to the fact that few publish failures, or that researchers continue to tweak or redesign the experiment until it in some way succeeds somewhat. Another aspect that is specific to the financial domain is that many test in “perfect” non-realistic environments. They do not take into account real aspects like brokerage fee, trading delays, volumes and difference in buy and sell prices. This way, many of the systems that in theory generate nice returns would fail in reality.

Chapter 3

Framework and System

This chapter presents the developed algorithmic trading framework used to implement candidate trading systems. In section 3.1 the framework in general terms is presented, focusing on its capabilities, thus giving a holistic perspective. The following section 3.2 expands on the initial overview, going into details about how the framework is implemented and what trade-offs are made. This leads up to chapter 4, which focuses on the actual trading systems that perform portfolio decisions.

3.1 Framework Overview

The framework is designed to allow for creation of a broad spectrum of trading systems. This is done by interconnecting computational processes, termed processor nodes, in modular structures. Although the framework has a generic designed and can handle a host of signal processing tasks, its main purpose is to facilitate the creation of trading and prediction systems. Trading systems are implemented as modules that interact with a set of supporting modules. A plethora of various nodes, each with a specialized purpose, have been implemented, resulting in a well-stocked proverbial tool chest. There is support for portfolio generation as well as other time series prediction, e.g., price and economic shock prediction.

With regard to the development of trading systems, as opposed to actual application of them, the following is a high-level description of how the data flow is configured using modules and processing nodes: Market data is extracted from one or more data sources. Then preprocessing and calculation of simple indicators follows, passing its results to a trader module, the embodiment of the various trading systems. These generate time dependent portfolios which are evaluated through a simulated market run and presented to the user as a textual or graphi-

cal output. All the stages above are housed by different modules from source and preprocessing modules to evaluation and sink modules.

Market data can be provided as a simple csv-file or downloaded from a data provider like Yahoo Finance API. Both current and historical data can be retrieved, though for this thesis' purposes only historical data are used.

Although the trading framework and systems employ extensive use of automation it is still vital to apply human intelligence in the setup phase. Apart from design and structural decisions human intelligence is required to get the most out of data. Preprocessing is as much of an art as it is a science and requires careful consideration. One supported approach to aid in this process is applying automated tuning as a port of data feature selection. A range of industry recognized indicators can be made available letting the system itself decide which it "wants" to use. This is achieved through an evolutionary process where input is decided based on trading system performance. When it comes to machine learning, too many features per datapoint is a recipe for overfitting and does not equate to better performance.

The trading systems can be implemented using any number of methods and strategies. Some are as simple as the follow-the-leader strategy, others use technical indicators. These are implemented as trivial framework capability tests.

More advanced, and academically accepted, methods include the efficient frontier (EF) portfolio from modern portfolio theory (MPT). It is implemented as processing node that solves the mean-variance quadratic optimization problem and thus finds the optimal portfolio. It does not solve the minimum lots version as this is NP-complete. The EF portfolio optimizer is used both as a benchmark and as a part of more complex trading systems. Both maximization of return with target risk and minimization of risk with target return are supported.

The challengers to the investment allocation scene are trading systems based on machine learning methods. The currently supported methods are:

1. Auto Regression (AR)
2. Feedforward Neural Network (FNN)
3. Elman Recurrent Neural Network (RNN)
4. General Regression Neural Network (GRNN)
5. Probabilistic Neural Network (PNN)
6. Support Vector Machine (SVM)
7. Support Vector Regression (SVR)

Most machine learning methods have some sort of parameters that are tuned by training. The framework divides this task into two levels. Low-level parameters are tuned by the machine learning method themselves. These are the parameters that are applied to the actual prediction task at hand. High-level meta-parameters define the trading systems structure and settings. Meta-parameters are tuned by either grid search or evolution depending on the solution space dependent on the solution space. Large solution spaces lend themselves better to evolutionary algorithms rather than blind grid search which might never finish.

The machine learning method's forté is their ability to predict, or generalize. This ability can be applied to price movements or risk measures and thus be a basis for generating a portfolio distribution. Generating portfolios from predictions is not trivial. There is the question of what actually to predict, e.g., do we predict price change in absolute terms, rate of change or just the direction? And when predictions are made, how do we convert them into a portfolio. Normalization of the predictions is a natural choice, yet this generates highly fluctuating portfolios that would incur huge transaction costs. This makes it important to apply measures to calm the traders down. On a first level this is taken into consideration by applying a transaction cost to simulated runs. Both fractional and constant transaction cost can be applied.

The transaction cost includes all costs associated with a trade, not only the brokerage fee. The Dow dataset does not include an ask-bid spreads thus making it hard to estimate achieved buy and sell prices. Because of this limitation, taking into account achievable buy/sell prices is not implemented directly, but rather through an additional fractional transaction charge.

One attempt to reduce the incurred transaction costs is through limiting the frequency of rebalancing. To control this a dedicated rebalancing node has been implemented. This is a node that evaluates whether or not it is favorable to rebalance the portfolio. In other words it decides if the trading system should commit to newly proposed portfolio. The framework supports two different criteria that can be used to judge when to rebalance. One is using a p-norm distance function such as the Manhattan distance between the portfolios, and the other using the Sharp ratio with zero risk. An example of simpler feature to mitigate potential losses is to apply stop-loss orders. Here the trading system defines a lower bound below the buy price at which to sell the position. This can also be set as a trailing stop, in which the lower bound is set at a fraction of the highest observed price since initial purchase.

The framework is not only built to facilitate trading systems, but also to find and evaluate them. In order to aid in this, a method for optimization of meta-

parameters was created. Both grid search and evolutionary optimization is implemented and can be used for analysis and tuning of different trading systems. Evaluation measures include, but are not limited to, the cumulative return; the rate of return mean, max, min and alpha; the risk measures variance, beta and Value-at-Risk; and the ratios Treynor, Sharp and Sortino. These are calculated at the end of a simulation for the entire simulation period, taking extra care to differentiate between training and test datasets.

The framework offers several different ways to present individual simulations. The most visual is the graph output of both the time dependent portfolio distribution, see figure 3.2, and the relative return, see figure 3.1. These are dynamic charts that animate the simulation development as it unveils. For a more detailed presentation all nodes can be outputted in a text-console windows sorted in a grid across the screen, see figure 3.3.

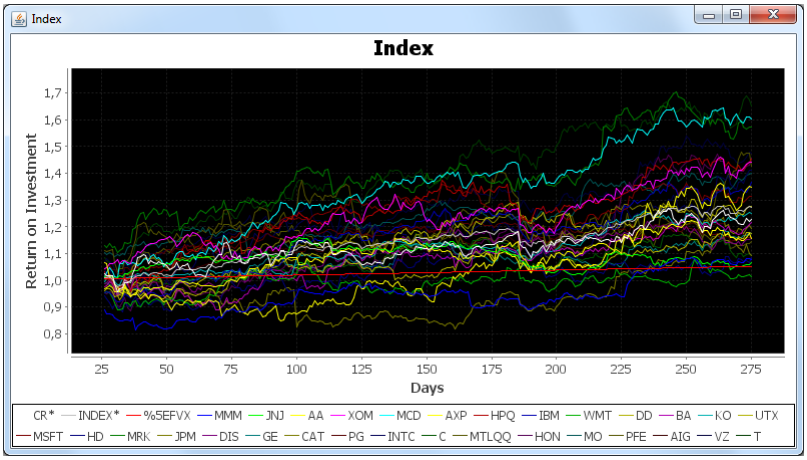


Figure 3.1: A screenshot of the line chart sink. It shows the relative prices of the Dow Jones Index (INDEX*) with its component companies in early 2006. Also included is the cumulative return of a short-term rolling mean-variance optimizing trading system (CR*) and US treasury bonds (%5RGVX).

In addition to the visualization of individual simulations, a graphical user interface (GUI) with integrated text-console allows for simpler user input and evaluation output. The GUI can be used to start multiple simulations at once in an effort to optimize meta-parameters. It does not present individual simulations as they run, but rather their evaluation results. All measures are calculated with transaction costs, expect the return measures and the variance which are calculated for both.

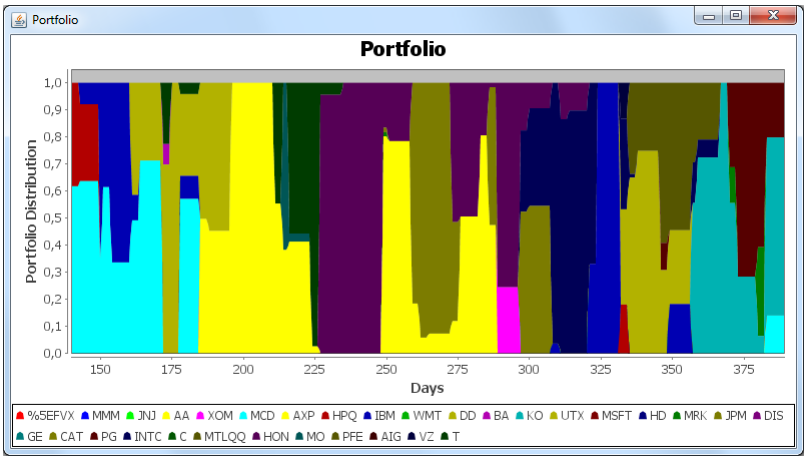


Figure 3.2: A screenshot of the stacked area chart sink. It shows the portfolio distribution from mid-2006 of the same simulation as in figure 3.1. The distribution is along the y-axis while time is along the x-axis.

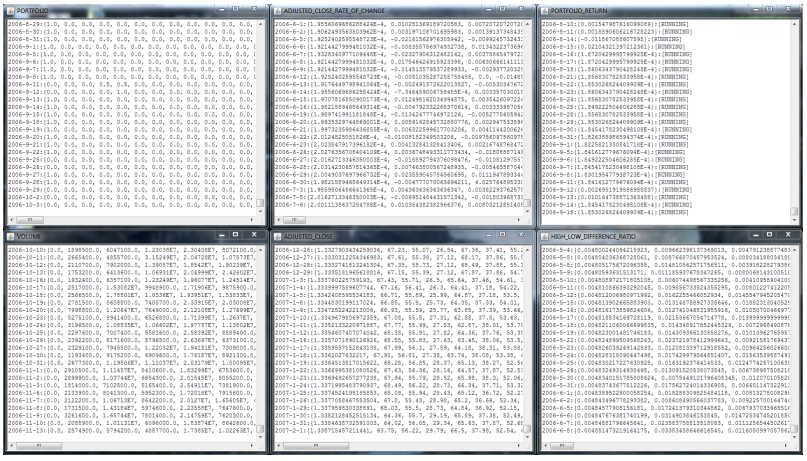


Figure 3.3: A screenshot of six text-consoles in a two-by-three grid. The text-consoles are monitoring various data streams. They can be hooked up to any stream in the data flow.

3.2 Framework Implementation

In essence, creating an automated trading system is a signal processing problem. Time series *signals* flow into a trading system and gets translated into portfolio distributions. From an engineering perspective the first question that comes to mind is how this should be done? Creating the system components in an ad hoc fashion and somehow combine them on the fly runs the risk of ending up with a jumbled incoherent mess of code, hindering reusability. Behind the curtains

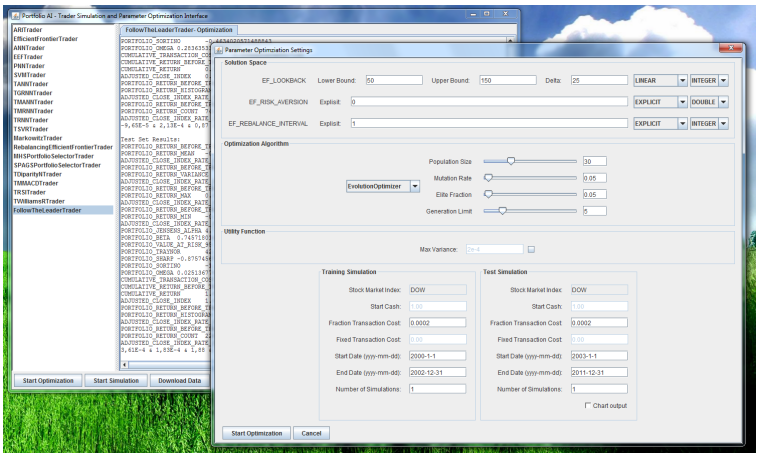


Figure 3.4: A screenshot of the simulation and optimization GUI for meta-parameter tuning. The background show a previous optimization run using a follow-the-leader trading system, while the foreground shows a optimization settings dialog of a mean-variance optimizing trader system.

of the trading systems presented in this paper there is a generic signal processing framework. It acts as the internal infrastructure for the flow and transformation of data. The framework is designed based on the principles of Kahn process networks (KPN), reviewed in section 2.3.1 on page 42.

The framework consists of a set of interconnected **processing nodes**, or processes in KPN jargon, that form a directed acyclic graph[1] (DAG). It is possible to implement support for cyclical networks, however, this introduces an unnecessary level of complexity it is better to avoid. The processing nodes are self-sufficient "miniature programs", or threads, that do operations on datapoints[2] in parallel. They house **processors** dedicated to manipulate and perform operations on datapoints. Each of these nodes contains a processor which performs the actual data processing. The different node types exist to place the processors in a larger context and direct the high level flow of data through the network. This abstraction is largely technical. In other words, the processing nodes give structure and aids in data flow, while the internal processors are the actual workhorse. It is important to note that no processor can exist without a node, and vice versa. Depending on the context *widget* node and *widget* processor will be referred to interchangeably.

[1]DAGs are flow networks that go in one direction only.
[2]Note that datapoints are vectors of real numbers.

3.2.1 Processing Nodes

Figure 3.5 illustrates the four basic types of processing nodes. **Source nodes** retrieves market data from external sources or generate them based on some given parameters. Their purpose is to serve as a starting point for the streams of datapoints that flow into the system. **Transit nodes** are versatile intermediary nodes used to perform operations on datapoints. There are many varied flavors of transit nodes, but in essence the differences are only semantics; they all take a datapoint as input, and output a processed datapoint. **Learning nodes** are processing nodes that are conceptually similar to transit nodes. In addition to having an input it also takes in ideal datapoints, i.e., labels. These ideal datapoints are used for supervised learning where a "correct" answer must be given for each input. This is done in hindsight, although in a on-line fashion. **Sink nodes** are the endpoint of a data stream. They can be used for presenting output to charts or text-consoles, or attached to a financial trading API for automated trading.

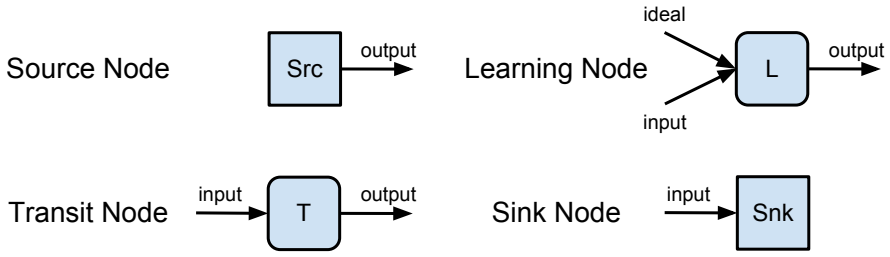


Figure 3.5: There are four basic processing nodes, which are used directly or in composite.

Processing nodes can be nested together into composite nodes, yielding more complex structures and operations. **Transit learning nodes** function as a learning node that has a leading preprocessing transit node. This is often useful when extending the functionality of a basic learner node. For instance, a linear regression learning node can be extended to an auto-regression learning node by simply applying a sliding window transit node first. **Transit pipelines** are a simple structure that strings multiple transit nodes together. This is useful for applying many operations that tend to go in sequence. **Transit collection nodes** are the most general of the composite nodes. It can take any DAG of transit nodes and run them as if its a single transit node.

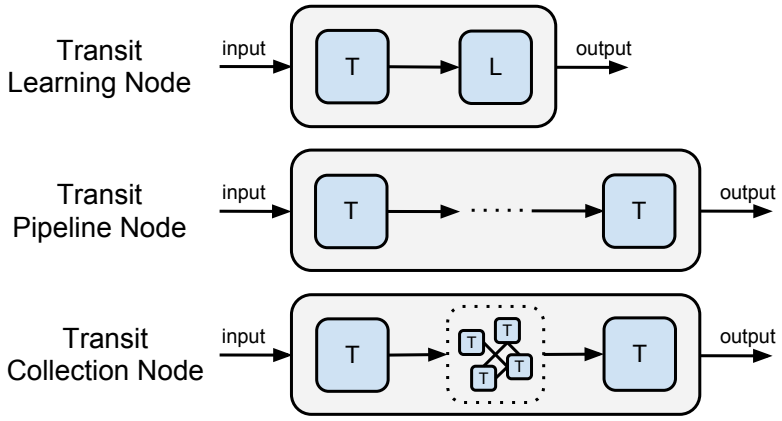


Figure 3.6: Composite nodes are complex nodes which are composed of multiple more basic nodes.

3.2.2 Subscription Based Data Flow Management

The proverbial glue that ties the different processing nodes together is the subscription queue. A subscription queue represents the edge[3] between two nodes. Each node has a publisher, subscriber or both. The **publisher** is responsible for pushing datapoints into the subscription queue. The datapoint sits in the queue until the consuming node's processor is finished processing its current datapoint and is ready to consume more data. It is the responsibility of the **subscriber** in the consuming node to retrieve data from the subscription queue and pass it to its processor. The subscription queue has in principle a limited capacity. If exceeded the publisher will halt processing in the node it is located in. This may create a cascading data backlog in upstream subscription queues and can in worst case result in a buffer overflow. Fortunately, for our datasets this is not a possibility.

The Many-to-Many Problem

Subscription between nodes is seldom one-to-one, more often it is many-to-many. The one-to-many case is handled by the publisher while the many-to-one case is handled by the subscriber. The publisher handles its problem by simply copying enough of its outgoing datapoints and passes them onto all outgoing subscription queues. The subscriber's problem is less simple since a consuming node doesn't always need the entire datapoint. It might want to discard some features, i.e., values, of a datapoint. In essence what the subscriber does is to retrieve all datapoints from the incoming subscription queues, discards some features, if any, and then merges the remnants into a single datapoint. The merged datapoint can then

[3]An edge in graph theory is the link, or connection, that make a path between two adjacent nodes.

finally get passed on to the processor of the node. After processing, the datapoint might be passed further downstream, if the node is a transit node; on the other hand if it's a sink node, it might be consumed by some external interface, e.g., a financial trading API or GUI.

Parallelization

The reason for using subscription based data flow, i.e., producer consumer design pattern, is to aid in parallelization of computation. This is possible because each node does not need to wait for the downstream nodes before it passes its processed datapoint on. It can put the result in the outgoing subscription queue, and carry on processing. Unfortunately, when starved for incoming datapoints it does need to wait until all of its upstream subscriptions are done processing. As with supply chain management, the system runs at the slowest node's speed.

Signaling

Because of the modular nature of processing nodes, it is not trivial to pass signals between them. The need for signaling is solved by piggybacking the signals on the datapoint stream. In order to do this datapoints are wrapped into **timepoints**. It is in fact the timepoint that actually flow through the network of nodes, while the datapoint and its signal are just along for the ride. The timepoints are so named because they, in addition to being datapoint and signal carriers, also keep track of the datapoint's associated timestamp. The available signals include `LOADING`, `RUNNING` and `DEAD`. The `LOADING` signal is used to indicate if the stream of data makes any sense yet. All processing nodes are required to send datapoints when they receive a datapoint, even if it does not have anything sensible to send, e.g., a learning node that is still collecting enough data to perform its first training round. This is required in order to keep the framework synchronized. When the `RUNNING` signal is flagged it means that the node can be sure that the datapoint it is receiving can be used. The final signal `DEAD`, of course, indicates that no more data is available. This can happen at the end of a trading day or when there is no more historical data to simulate on. When the `DEAD` signal is received, the node finishes processing the last datapoint and shuts down.

3.2.3 Processors

The following is a short introduction to the many processors of the framework. Because of the nature of the framework, an innumerable number of processors have been implemented. All main categories are covered, but only the most important and a few examples of trivial, implementations of processors will be covered.

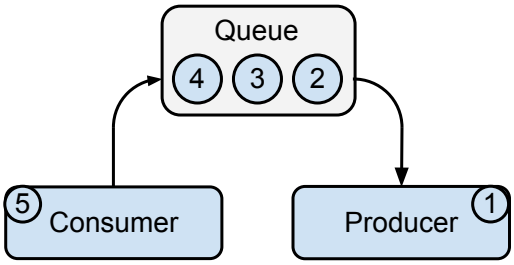


Figure 3.7: The classic consumer-producer problem can be solved using an intermediary queue that temporarily stores semi-finished "produce".

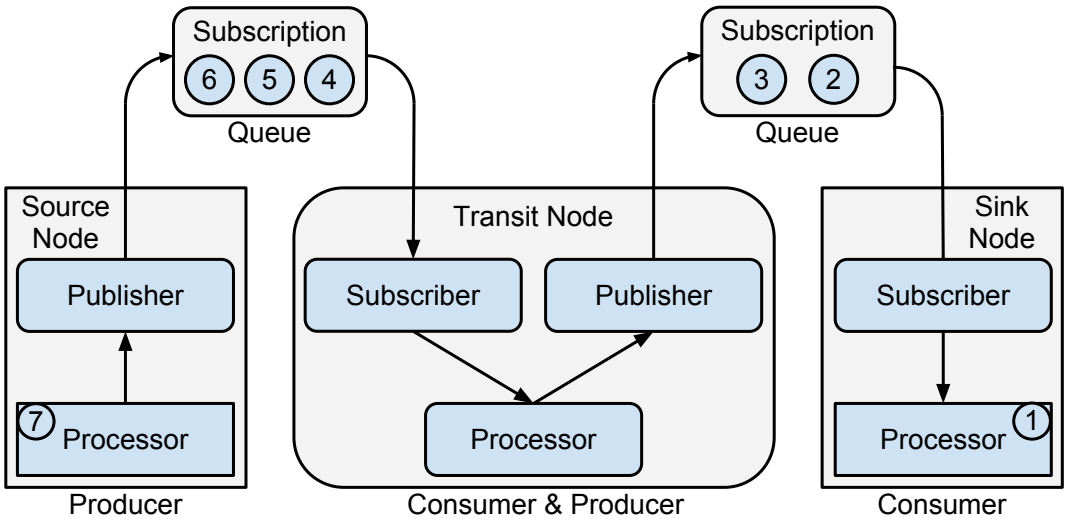


Figure 3.8: This figure shows how datapoints can flow from a source node to a sink node using subscription based data flow management. Note that the datapoints may be vectors, but are single values in this illustration.

Source Processors

As earlier described, source processors retrieve market data from external sources, or generate them based on some given parameters. The sources may be very different, and combined by the interconnection of nodes, making it easy to combine data from several sources simultaneously. This endows the trading systems' with the ability to access different, and maybe unconventional data sources in order to achieve a competitive advantage relative to traditional systems. Though, because of scope, this is used in a limited fashion in this thesis. More on this in future work, section 6.5.

Generating Test and Dummy Data. For testing purposes, dummy data is needed to make sure that nodes are acting in accordance with expectations. If real market data is used you cannot always be sure if bad results are due to the method's inherent traits or if it is due to a flawed implementation. Dummy source processors that produce dummy data can be invaluable to debugging. A classic example of such a source processor is the **trigonometry source processor**. It takes trigonometric functions, e.g. the sine or cosine function, and transforms it based on desired amplitude, mean, period, lifetime and time shift. You can always expect a good prediction method to be able to predict future movements of these functions based on history.

Real-Time Market Data. Only source processors using historical market data have been fully implemented. This is due to the proprietary nature and low resolution of available real-time market data.

Historic Market Data. For evaluation of trading systems real market data is required. There are multiple ways to retrieve this from third party sources. The framework supports retrieval of data from the financial Yahoo API and by reading cvs-files directly. Market data have structures that is most often represented in a per-trading-instrument fashion, i.e., each trading instrument has the ubiquitous features open, high, low, close, volume and adjusted close. Before processing these are split into different datasets, one for each feature. This is illustrated in figure 3.9. In other words datapoints that flow out of each source node is a collection of similar features, e.g., only volume or close features. This makes subscription to one kind of feature possible.

Missing Market Data. Another concern is that some trading instruments might have missing data. This is especially true for trading instruments that are taken off or put on the index in question. Sporadically missing market data is handled by simple deletion of all temporally corresponding data entries; they never achieve datapoint status. Introduction and expulsion of trading instruments is handled by always using the most recent index composition at the start of a simulation and not changing it, even if the index officially changes the composition during the simulation.

Transit Processors

As the name implies, the transit processors are intermediary processors. These perform most processing tasks except learning.

Transit nodes are the most common nodes, a complete list can be found in the appendix 7. Transit nodes are for the most part conventional methods and techniques, the most advanced one is a quadratic optimizing algorithm to find Markowitz

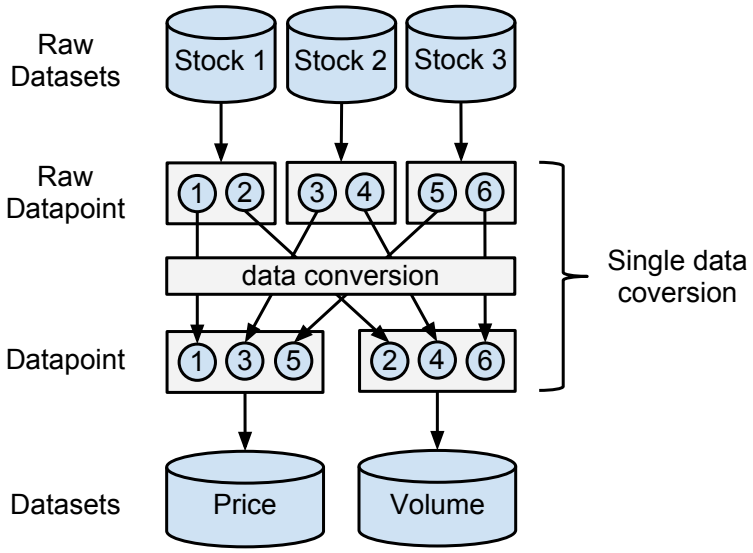


Figure 3.9: Raw market data is stored in a per-trading-instrument fashion and coverted to datapoints that have similar features, like price and volume.

portfolios. There are four main categories of transit processors.

Operator Transit Processors. All processing of data needs to be done inside nodes in this system. This includes simple arithmetic operations like sum and dot product. Most of these are rather trivial and not important to discuss further. Filter processors like moving average are also considered operators.

Portfolio Transit Processors. The trading systems' end product is a portfolio. These are most often produced by a learning processor, but several transit processors also handle portfolios. These generally have one of two purposes. One kind produces their own portfolios while the other kind modifies existing ones. The portfolio producers are often simpler technical analysis techniques for creating portfolios that does not require any learning. The modifiers are made to ensure correct portfolios as well as improving on existing ones in other fashions. Stop loss is for instance implemented as a portfolio transit processor as well as the processor deciding when to rebalance.

Meta Transit Processors. A set of operators operate on other datapoints at a higher level. These are called meta transit processor as they are usually set to operate on already existing transit processors. The defining example of this is the window transit processor which takes an output stream from one processor and chunks the datapoints together in time windows of a certain size to allow the next agent to get a time context on its input. Other finds and supplies the time context

of a datapoint and one synch different data streams.

Risk Evaluators. Risk evaluators are a set of transit processors that calculates common risk measures. Examples of the implemented evaluators are Sharpe, Sortino, Variance, Covariance, Beta and value at risk. These are used both in prediction and for validation of the results in order to incorporate and evaluate risk.

Learning Processors

Learning Nodes are processing nodes that are conceptually similar to transit nodes. In addition to having an input it also takes in ideal datapoints, i.e., labels. These ideal datapoints are used for supervised learning where a "correct" answer must be given for each input. This is done in hindsight, although in a on-line fashion.

Prediction processors are more advanced processors with learning, contained in prediction nodes. These are by far the most technically advanced processors, and an important aspect of this thesis is to test their contribution to the trading systems. For reasons of modifiability and due to the limited time scope most are implemented with external frameworks and the injected into the framework. This way the implementation can be changed without affecting the rest of the system. The common denominator across all prediction processors is their learning capabilities. Most learn in quite different ways, but all needs to be trained to function efficiently. Most of this training is supervised and the prediction processors must be provided results or other labels together with their normal input. This is handled the same way as normal input aside from using a different method in the connection process. All training is scheduled by a training schedule implementing a common interface to decide when to train. This same schedule defines when evolution is run. The central predictors are linked to the central technology methods:

A collection of nodes are fitted together to form trading systems. These trading systems are the end results that form autonomous traders capable of giving portfolio advice on their own. When connected to an actual market these will be capable of autonomous trading. The framework was built to allow many such trading systems to be evaluated in order to facilitate a search for well performing ones.

An example of a trading system implemented with nodes is the efficient frontier optimizer. This is illustrated in figure 3.10.

Historical stock price data are sent through a **window processor** and then a **rate of return processor** to acquire the expected return. The expected return are then sent to the **Markowitz processor** and a **covariance processor**. The

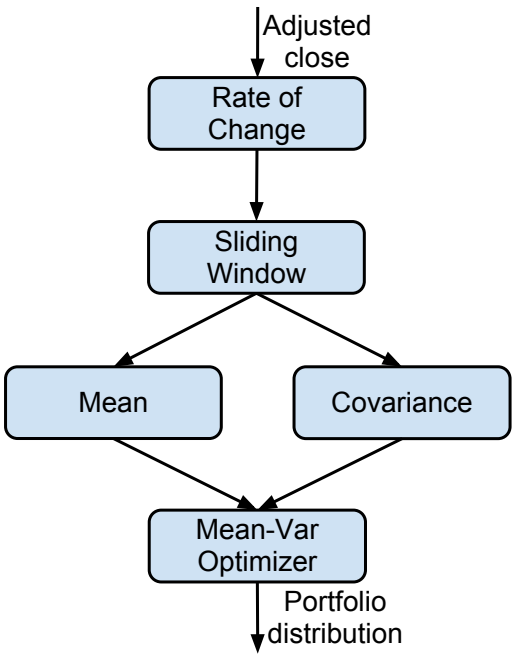


Figure 3.10: Mean-variance optimizing trading system.

output from the covariance is also sent to the Markowitz processor and it can then output efficient portfolios. These are then sent to a **rebalancer** which decides whether it is profitable to exchange the current portfolio for the new one. The rebalancer sends the chosen portfolio to a **finalizer** which removes small positions and then normalizes the portfolio again. At last this final portfolio is sent to some sink to output the results to the user. This can be textual or graphed. At a parallel the cumulative return of this portfolio can be measured by including more processors, thus effectively evaluating the performance of the Markowitz portfolio trader. This is not shown in the figure

Auto Regression. The simplest learning processor implemented is the auto regression. Coupled with a moving average this can implement the classical ARIMA predictor. This is a standard benchmark for any time series prediction. Auto regression is not implemented for its regression capabilities, but more to be a lower bound in which to compare other classifiers to as a sanity check. Auto regression is implemented with the aid of the Weka framework [76].

Artificial Neural Network. There are mainly two categories of ANNs at use in this system. The standard multilayered perceptron neural networks (MLPNN) and the more advanced radial basis function (RBF) networks. Of the standard version a common feed-forward network is used as well as a recurrent version of this, they are

called feedforward neural network (FNN) and recurrent neural networks (RNN). There are also two different RBF networks, the general regression neural network (GRNN) and the probabilistic neural network (PNN). The implementation is again very similar aside from the two last layers. In essence the GRNN is a regression variant of the PNN, which only does classification. All ANNs are implemented with the help of the Encog framework [81]. For an illustration of the relations between the ANNs see figure 2.16.

The ANNs have two primarily uses in this system. RNN and GRNN are used for predictions of prices and other data as they have shown great promise at this in other papers [98; 124]. The different ANNs are also used to create portfolio distributions. This is a more experimental use, and a central hypothesis of this paper is whether such a technique is feasible.

The multilayered perceptron networks are trained using resilient propagation[84; 120]. This is the recommended one in the Encog framework documentation[81] and informal tests against other prominent techniques like scaled conjugate gradient demonstrated its superiority. The GRNN and PNN are trained with a conjugate gradient algorithm, much based on algorithms from Masters [102].

In addition to the resilient propagation the ANN can employ evolution to adjust its topology. The phenotype defines the number of layers and neurons inside the layers. Each integer is one hidden layer and its value defines the number of neurons in it. The input and output layer comes in addition to these. Thus `3,6,3` means three hidden layers with respectively 3, 6 and 3 neurons. By running evolution at an interval the ANNs thus achieve a dynamic architecture that can change in complexity by need. To avoid unnecessary complexity the genotypes are given a size penalty giving simpler networks a benefit.

Support Vector Machines. There are two SVM implementations, much like with the RBF networks, one for standard for classification and an support vector regression (SVR) implementation. Suport Vector Machines are also implemented through the Encog Framework [81] Support Vector Machines (SVM) are used in much the same way as ANNs as time series predictors. SVM have shown strong results in applications of financial time series prediction [126], too much to be ignored in this project. SVMs are also combined with ANNs as several other studies done this and gotten very good results [108; 115; 133].

The Encog implementation of SVM is based on libsvm [41]. This uses an intelligent grid search to set the parameters of the SVM [61].

Sink Processors

Sink Nodes are the endpoint of a data stream. They can be used for presenting output to charts or text-consoles, or attached to a financial trading API for automated trading.

The scope of this project is to test a concept; such an experiment would not be feasible to perform on a live market. This is the reason why a market simulator was created and why connecting the system to a real trading were not a priority. Still it was important to make the framework flexible and easily modifiable on how and where the output where to be sent. In this project most sinks made were to present the data in a form convenient for the experiments that were run. For future work connector sink to other systems either trading software or expert systems would be relevant and easily possible.

3.2.4 Modules

There are many nodes and it is cumbersome to create and link all the required ones to achieve some functionality. In order to abstract such functions away modules were created. A module is a collection of nodes that takes as input a predefined set of parameters and produces another set of parameters. There are modules corresponding to the same prefixes the nodes have representing whether they produce, handle or output data.

Source Module

Source modules retrieve and offer data from a certain source. The data taken from a source often needs some processing before it can be effectively used by other parts of the system. This can be taken care of by the module so only one module for each source has to be made.

Trader Module

The source and sink modules can usually be shared among all trading systems. The point of variation is then captured in the trading module, which then becomes the core of the trading system. The different trading systems are implemented as trader modules and any difference in output or input are handled by configurations.

Prediction Module

The prediction module is similar to the trading module except that it contains configurations that predict single stocks instead of whole portfolios. This is mostly used for experiments with predictors in this project, but it could also be used for single stock trading independent of portfolio considerations.

Sink Module

Same as with the source module the way to output the result is done through a generic module, the Sink module. If the framework is to be connected to real markets the connection adapter to the broker software would be a sink module.

3.2.5 Module Architectures

These modules can be connected in different ways to form module architectures. The most important one at this stage is the market evaluation module architecture allowing for simulation of a real market and evaluation based on this. In the case of a live system this would be exchanged for architecture with the required live source modules and the necessary connections back to the market through some sink modules, possibly with the option of human intervention through an interface. Here the current market simulating architecture is discussed.

Market Simulation

The framework as of now is connected to two datasets and capable of simulating market activity on both of these. These are the stocks from the Dow index from 2000 to 2012 and from the OBX index from 2006 to 2012. It handles a resolution down to daily trades and both fixed and relative transaction costs. In the case of the Dow stocks the company composition is dynamically changed to the companies the index consisted of. In the case of OBX all data is taken from the stocks the index comprise of in February 2012. More on the exact simulation setup is described in the experiments section 5.1.

Market Application

The system lacks some functionality before it is ready to run live. As of now it has a semi-live runtime input data interface against Yahoo, this is however not stable enough to risk as a live stock trader. The trading systems needs a real data stream, however as they only require daily data this does not need to be very expensive, depending on the amount of data required. The simulations done ran on very limited data, but the systems may perform better on a larger and more diverse data foundation.

The systems further need a connection to the markets or some sort of broker software to be able to post orders. Examples of possible solutions are Ninja Trader and Trade Station. The integration with such a system will be trivial due to the modular nature of the current system.

All in all there are some costs with expanding this to a live system, but they are fairly small.

3.2.6 Learning Scheme

The financial markets are volatile and ever changing environments. To be able to perform well in such an environment the trading systems must be able to adapt and a good way to achieve this adaptation is learning. Learning also have other uses, for instance as a search algorithm when the search space is vast or complex. In this system the learning are separated in two main types. Meta-parameter optimization is the search for the optimal combination of a set parameters that otherwise remain static in the system. Sub-parameter learning is adaptation of parameters in an online setting as the trading system runs, with the aim to continually adapt to changing conditions.

Meta-parameter Optimization

Meta-parameters are variables defining the conditions for a complete simulation of a trading system. Typical examples are the training set size, window sizes and various thresholds. Finding the optimal or at least a good fit for these is an important part of the tuning the trading systems. There are many of these parameters and they vary depending on the specific trading system. In order to efficiently and objectively search tune the meta-parameters a framework were built to support general optimization of any trading system. This framework is operational from the user interface. Here one can select a trading system and the scope of the relevant variables and then choose between a grid search and an evolutionary search for the best combination of parameters. The evaluation is done in the same way as the experiments with a set of simulated runs in which the average performance are calculated. Evolution is implemented with the aid of the Watchmaker framework [49].

Sub-parameter Learning

Sub-parameter learning is the common denominator of the learning nodes in the trading systems. These are by far the most complex pieces of the trading systems and largely define the systems they inhabit. These are mostly classifiers that require continual learning in order to perform on a high level, but done right these outperform most static classifiers and they are far more adaptive to changing circumstances. The training is controlled by a training schedule that notifies the processors when they are to train. Training is done over a set of datapoints which length is a meta-parameter. The training interval is also a meta-parameter. The training window is a sliding window trailing behind the current point in time at which the trading system has reached. The actual training algorithm used by the node is dependent on the classifier as described in an earlier section.

3.2.7 Framework Limitations

The system design is far from flawless, in this section it will be elaborated on some of the limitations and potential weaknesses the system is predicted to have.

Input Complexity. One of the main issues expected with the system is the performance. There are many inputs, and many of these need one value for each asset as they are asset dependent. This makes for a large amount of inputs which leads to big and complex ANNs. All this may limit the number of stocks the system can analyze, thus it may lose out on arbitrage opportunities. There is also a limit on the number of input the system can be given, and the reduction in inputs might lead the system to miss out on patterns it could have exploited.

Black Box. Another issue is the black-box nature of this system. There is no good way of knowing why most agents does the choices they do. This is much of the reason to why evaluation of robustness and adaptivity is introduced so it can be shown statistically that the system still performs in a stable manner.

Not Built for Speed. The system is not built to take on HFT systems. It is designed for concurrency and to be run on clusters, but it will never be able to compete with the systems striving for speed. Aside from the hardware such speeds require very simple algorithms implemented as far down in the system as possible, if not hard wired, not much beyond that. What would be possible to achieve both the adaptivity and intelligence of this system and the speed of HFT systems would be to use evolution and other classification techniques in order to discover patterns and then implement simple rules to exploit these. The pattern discovery process could be run in market downtime. This is anyhow far beyond any scope of this project, and have many challenges. More on this idea in future work.

One must still remember that this is a semi-online trading system with relearning phases. A natural relearning schedule would be to perform meta-parameter optimization on weekends and sub-parameter relearning on a daily basis. This utilizes market downtime periods which are ideal times for the heavy computation required for relearning. Shorter periods could be used with hourly sub-parameter relearning and daily meta-parameter optimization, although this requires running at least two sets of systems in parallel. One learning while the other is running, and hot-swapping at regular intervals. An alternative to scheduling is the use of the previously mentioned hedge algorithm which is an online learning algorithm. This reduces the problem to only low level relearning.

Not Connected to Real Markets. The system is as of now still only operating on a simulated market. It is however build to easily be able to connect to a market API, but this would give no real academic value as it is no good to experiment on

live market as the results would not be reproducible, and it would be costly.

Trade Execution Algorithms. In today's world of high frequency trading an actor that naively posts a buy or sell signal will be manipulated into receiving a sub-optimal price by other actors that will undercut it. To effectively handle trades in an electronic market today one needs a specific algorithm for how to post a series of strategic signals in order to identify and achieve the best possible price. This system has no such algorithm as this aspect were not a part of the scope, this is however a crucial part that needs to be done before any real life application of the system can be run with any success.

Lack of Long Term Memory. The system has two layers of learning. Evolution sets long term parameters while short term learning like SVM or ANN learns on a day to day basis. This does not allow for very long memory, as the meta parameters to a small degree can recognize patterns and the short term learning is usually at most a 100 days. The length of this training was however found through evolution and one can argue that if it were appropriate to have a longer learning set this would have been shown through evolution. As such there is at least some claim to say that if there were important patterns longer than 100 days the system have had its chance to find these and chosen to focus on shorter patterns.

No shorting. This system does not support shorting of stocks. It is generally limited to two signals. Buy and not buy, and then a portfolio is created by the buying the stocks recommended. As such there is no need for a sell signal as any stocks without recommendation are sold if they were held at that point. However if the system predicts a downturn the system has no way of capitalize on this aside from avoiding the potential loss. While clearly reducing the profitability of the system this also reduces the risk as shorting in theory have an unlimited loss potential. In addition there are stricter rules for shorting and several markets that do not allow it. This would further complicate an eventual implementation. In the scope of this project this was not seen as a priority. The profitability shown in the long buying capabilities of the system is seen as enough evidence of the viability of the system.

Chapter 4

Trading Systems

This chapter reviews the various candidate trading system architectures, designs and implementations. Section 4.1 looks at the alternative structures and strategies which a trading system can be endowed with. The section is dedicated to the general trader system architectures and strategies that have shown promise. Section 4.2 reviews the actual candidate systems. In section 5.3 these are evaluated under fair simulated conditions, taking extra precautions to detect overfitting.

4.1 Trading System Implementation

There are many architecturally similar trading systems that employ very different techniques. Common to them all is their **portfolio generator** and **rebalancer**. The former creates portfolios based on the applied techniques, while the latter makes a decision of whether or not to commit to the suggested portfolio. The basic trading system is illustrated in figure 4.1.

4.1.1 Portfolio Generation Strategies

There are two fundamentally different ways to generate portfolios, each lending its support to one of the two main paradigms in financial theory. The first hinges on modern portfolio theory and its efficient market hypothesis. The approach accepts the unpredictable stochasticity of markets, and suggests using diversification methods, like mean-variance optimization, to create efficient portfolios. In practice this result in a set of trading systems called **efficient frontier traders** . The alternative hinges on the behavioral finance paradigm, where an attempt at uncovering market inefficiencies give rise to **price prediction traders**. These make price movement predictions based on market developments. Both approaches yield

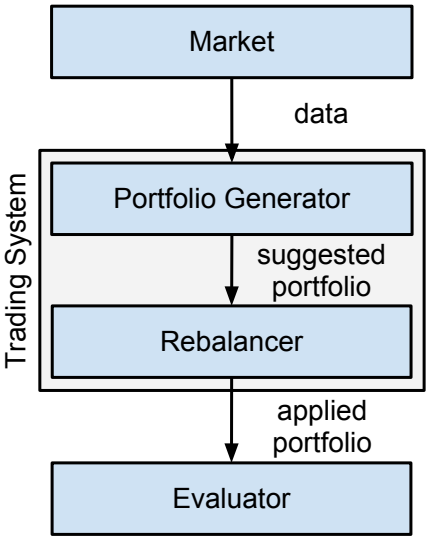


Figure 4.1: The most basic trader has a portfolio generator that produces portfolios based on market data and a rebalancer that figures out if it is a good idea to use this portfolio.

very different trader designs. A third more exotic type of trading system termed **portfolio selection traders** also exist. They try to select or combine portfolios generated by several basic portfolio generators, i.e., miniature trading systems. The following sections go into detail on these main portfolio generation strategies.

Price Prediction Traders

As illustrated in figure 4.2, there are three main architectures used to perform price movement prediction. The simplest **one-to-one predictors** employ methods that try to predict price movements individually based on only historical price development. The second **many-to-one predictors** uses methods that try to predict individual price movements using all available market data. The final **many-to-many predictors** apply the most general methods; these can, in one go, make multiple price predictions based on all available market data. As figure 4.3 illustrates, the more general prediction methods can perform the task of the simpler ones.

One-to-one Architecture. The most basic price predictors apply simple operations on the data it receives. The one-to-one price prediction architecture looks solely on the price development of an asset and tries to predict its next movement. Admittedly this is a highly trivial ordeal. The traders that employ this architecture are mainly **technical analysis traders**. These are the most basic traders

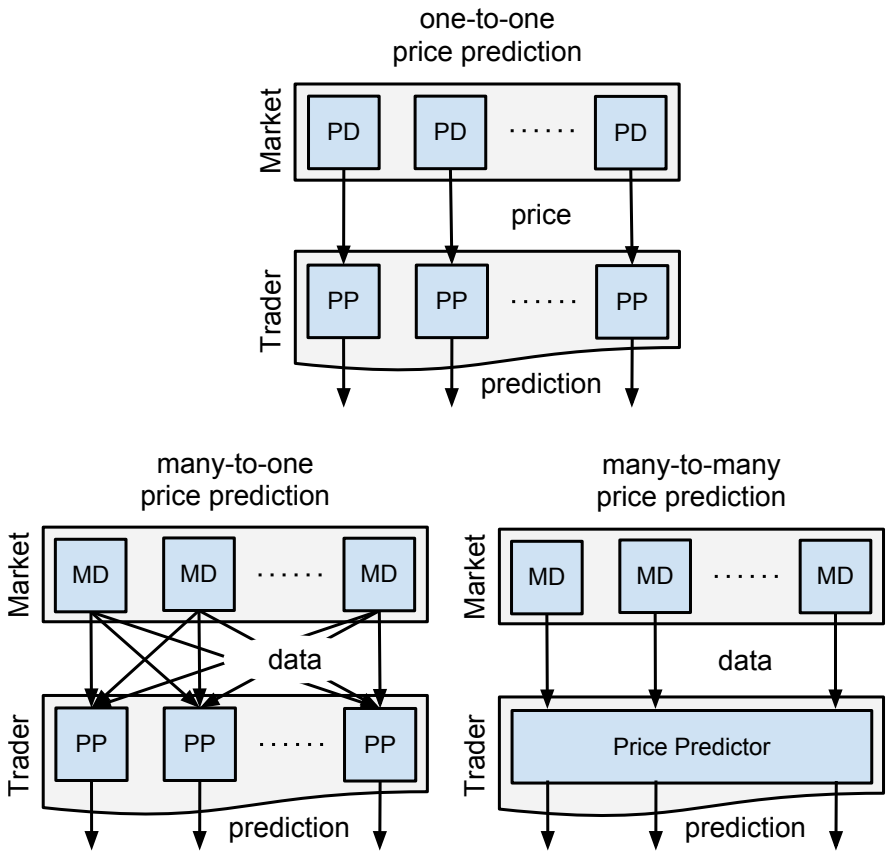


Figure 4.2: There are three main methods for price prediction. The first sends price data (PD) to individual price predictors (PP) that do not use any other information to predict data. The second employs multiple price predictors using all available market data (MD). The final method uses a single predictor that tries to simultaneously predict all price movements given all available market data.

and apply only the most rudimentary filters in an effort to predict future price development. The indicators these use are described in section 2.2. A glaring problem is that these indicators are all lagging, and have no learning.

Many-to-one Architecture. A more ambitious approach is the many-to-one price prediction architecture. It looks at all available data of the market and tries to predict individual price changes. All traders that do this are learning traders, i.e., they use learning nodes capable of making generalizations and find patterns. This architecture requires many of these learning nodes, one for each asset. The collection of these learning nodes forwards their predictions to a mechanism for

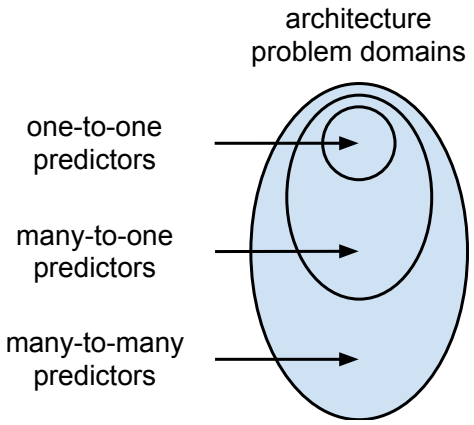


Figure 4.3: The more general prediction architectures can perform the task of the simpler once.

creating portfolios. An example of a trader using the many-to-one architecture is the SVM trader. It is able to output only two classes, i.e., values, either 0 or 1. It is applied as a price movement indicator.

Many-to-many Architecture. There are few price prediction methods that handle multiple inputs and multiple outputs. In this thesis, only the neural network methods are capable of doing this, and actually be able to make use of the inherent properties of considering all data at once. The traders that apply the many-to-many price prediction architecture are able to process any data and output predictions. The most notable explicit example of such a method is the MLPNN. By presenting price development together with the preceding input it is able to make a prediction of which way the market might move.

An important concept to understand about the price predictors is that the many-to-one and many-to-many architectures can use any data as a basis for price prediction, not only the price itself.

Converting Price Predictions to Portfolios. Since price predictions can't be used directly as a portfolio, a conversion process must be performed. Some output a binary vector of 0s and 1s or a decimal in between, either indicating to what degree the price predictor believes the price will move. There are many ways to transform such predictions into true portfolios.

The main method for converting price indicators $\boldsymbol{v}$ to a portfolio $\boldsymbol{w}$ is by **normalizing** it. In essence what is done is to convert the price indicator vector $\boldsymbol{v} \in [0, 1]^n$ into a simplex where the $\mathbf{1}^{\mathrm{T}}\boldsymbol{w} = 1$ and $\boldsymbol{w} \geq 0$. In this context normalization

simply applies the transformation of 4.1.

$$\boldsymbol{w} = \frac{\boldsymbol{v}}{\mathbf{1}^{\mathrm{T}}\boldsymbol{v}} \tag{4.1}$$

A special case is when $\mathbf{1}^{\mathrm{T}}\boldsymbol{v} = 0$. This can happen when the price predictor does not believe in any asset. This is solved by setting the portfolio to a fail-safe portfolio. Typically this is the uniform portfolio $\boldsymbol{w} = \mathbf{1}/n$ or the risk-free asset explicitly. Because of rounding errors all trading systems are required to perform normalization at the end to ensure compliance to the portfolio constraints.

Price movement indication is most often within the interval $[0, 1]$ where 0 is a full commitment to the belief in a downward movement and 1 is the opposite belief in an upward movement. However this is not always the case. Sometimes, as with the MACD histogram, it might be a real number oscillating around zero. Often it might be useful to either force the indicator to be in the mentioned interval, e.g., by using a sigmoid function. It can even be useful to force the indicator to be $\{0, 1\}$ exactly, i.e., force it to be either 0 or 1. A way to do this is by applying a tuneable threshold parameter γ using a indicator function $I(p > \gamma)$, where p can be any indicator. It can even be beneficial to do this to already conforming indicators that are within the $[0, 1]$ interval. It can make portfolio creation through normalization less variable thus reducing transaction cost.

After normalization many finalization steps might be performed. Insignificant positions might be removed, or even just the top positions kept. More on this in section 4.1.2.

Efficient Frontier Traders

The efficient frontier traders use modern portfolio theory as a foundation. It consists of a node that calculates the mean-variance optimal portfolio given expected return and the covariance matrix. These data are created from the historical data. The length of the history is set by a meta-parameter. As described in 2.2, a target return, variance, or risk aversion is supplied as a parameter.

Finding an efficient portfolio with the minimum transaction lot is a NP-complete problem[99]. To find an efficient portfolio given either a variance, return or risk aversion factor is a convex problem, easily solved by many pre-existing packages. This system uses ojAlgo[3], an open source java framework for mathematics and optimization; it is especially suited for the financial domain.

Portfolio Selection Traders

The final portfolio generation approach applies ideas from **prediction with expert advice**[40]. The idea is to let different experts make advice, i.e., let portfolio generators make portfolios, and then, to various degrees, listen to one or more of them. The classical interpretation is that the portfolio selector allots a portion of its available investment capital to each expert based on how much it believes in them.

In mathematical terms. Let expert k, of a total of K experts, output the portfolio $\boldsymbol{w}_k$, then a portfolio matrix $\mathbf{W}$, consisting of K column vectors $\boldsymbol{w}_k \in [0,1]^n$, can represent all the advice given for a period. The portfolio selector, then allocates commitment, or funds, $z_k \in [0,1]$ to each expert k. The collective allotment $\boldsymbol{z} = [z_1 \ \cdots \ z_K]^{\mathrm{T}}$, constrained by $\mathbf{1}^{\mathrm{T}}\boldsymbol{z} = 1$, can thus be applied to the portfolio advice matrix $\mathbf{W}$ to yield a trading system output portfolio $\boldsymbol{w} \in [0,1]^n$, as shown in (4.2). In essence, the outputted portfolio is a convex combination of the expert advice. That is, the constraints on $\boldsymbol{z}$ and $\boldsymbol{w}_k$ guarantees that portfolio $\boldsymbol{w}$ is in the convex hull of all $\boldsymbol{w}_k$ and that it fulfills the portfolio simplex requirements.

$$\boldsymbol{w} = \mathbf{W}\boldsymbol{z} = \boldsymbol{w}_1 z_1 + \cdots + \boldsymbol{w}_K z_K \tag{4.2}$$

A problem with this approach is that it does not explicitly take into consideration the expert advice portfolio's internal risk-return characteristics. What might have been a good portfolio due to its inherent traits might be lost when combined.

4.1.2 Rebalancing and Finalization Strategies

It is not always appropriate to directly apply a portfolio from a portfolio generator. Applying a portfolio with low or no expected increase in return or decrease in risk can be bad because of transaction costs. Investing in positions that have negligible size reduces stability and increase administration. Making sure a portfolio conforms to requirements and that the suggested portfolio is a good idea is the domain of the rebalancer and the process of finalizing the portfolio.

Rebalancing

In a practical sense, the rebalancer is a transit processor that inputs a stream of portfolios and decides when it is appropriate to **rebalance**. When a portfolio is rebalanced it is in essence applied to the market incurring a transaction cost. The main reason for having a rebalancer is the potential for astronomical transaction costs when a portfolio generator is left to its own accord. There are different rebalancer implementations. The two main implementations is the **portfolio distance rebalancer** and the **expected portfolio utility rebalancer** .

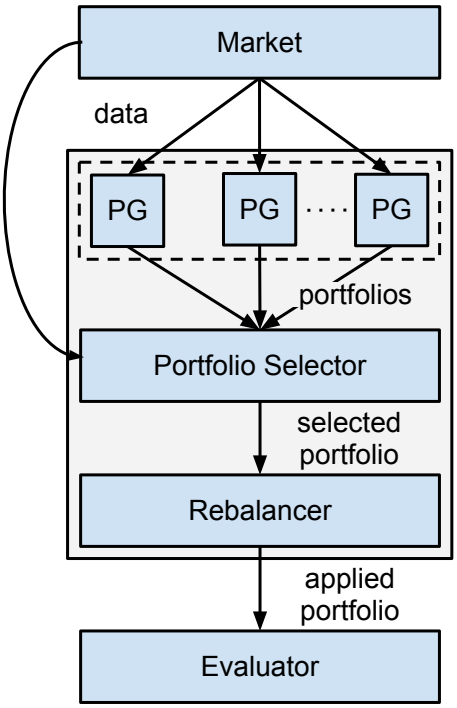


Figure 4.4: A more advanced trading system that applies multiple portfolio generators (PG), aggregating them by using a portfolio selector. A single or a mixture of multiple portfolios is selected for rebalancing review.

Portfolio Distance Rebalancer. The simpler portfolio distance rebalancer calculates the p-norm distance function between the existing portfolio $\boldsymbol{w}_{t-1}$ and the new suggested portfolio $\boldsymbol{u}_t$. If the portfolio distance is larger than some absolute term γ then it may be applied as shown in 4.3.

$$\boldsymbol{w}_t = \begin{cases} ||\boldsymbol{u}_t - \boldsymbol{w}_{t-1}||_p > \gamma & \boldsymbol{u}_t \\ \text{else} & \boldsymbol{w}_{t-1} \end{cases} \tag{4.3}$$

A typical norm to use is $p = 1$, i.e., the Manhattan distance. This method simple and quick, but can lead to frequent rebalancing if the input oscillates between two portfolios with very different distribution, but with quite similar financial properties i.e. mean and variance.

Expected Portfolio Utility Rebalancer. As discussed in section 2.1.3, utility functions can be used to numerically evaluate the utility of a portfolio based on its performance measures. It is possible to use a utility function to evaluate both

the existing portfolio compared to a new suggested portfolio. If the new portfolio is better by some margin then it is applied. There is two ways to use a threshold γ to achieve this. The first is similar to that of (4.3) and looks at how much better in absolute terms a portfolio's utility must be. This approach is shown in (4.4).

$$\boldsymbol{w}_t = \begin{cases} U(\boldsymbol{u}_t; \mathbf{R}) - U(\boldsymbol{w}_t; \mathbf{R}) > \gamma & \boldsymbol{u}_t \\ \text{else} & \boldsymbol{w}_{t-1} \end{cases} \tag{4.4}$$

The problem here is that it isn't always simple to know how much better, in utility units, a portfolio should be. Granted it can be evolved as a meta-parameter thus a useful threshold will emerge. An alternative that is more intuitive to humans is to specificity in relative terms how much better the portfolio must be. This is shown in (4.5), where it states that a new suggested portfolio should be used if its utility is γ percent better than the existing portfolio's utility.

$$\boldsymbol{w_t} = \begin{cases} U(\boldsymbol{w_t}; \mathbf{R}) > (1+\gamma) U(\boldsymbol{u_t}; \mathbf{R}) & \boldsymbol{u}_t \\ \text{else} & \boldsymbol{w}_{t-1} \end{cases} \tag{4.5}$$

Notice that the utility function has return matrix $\mathbf{R}$ as a known parameter. This is the sample returns that are used to calculate the utility. It could be the entire return matrix, or just a recent subset. Two utility functions have been applied. The first (4.6) is a linear combination of the mean return and the variance, using a risk aversion factor ρ, time periods T, and definition (2.5) of the covariance matrix $\boldsymbol{\Sigma}$.

$$U(\boldsymbol{w}; \mathbf{R}, \rho) = \bar{r}_{\boldsymbol{w}} - \rho \sigma_{\boldsymbol{w}}^2 = \frac{\mathbf{1}^{\mathrm{T}} \mathbf{R} \boldsymbol{w}}{T} - \rho \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{w} \tag{4.6}$$

An alternative utility uses the risk-return ratio as shown in (4.7). This version is only applicable for datasets without a risk-free asset. By allocating all investment capital to the risk-free asset the risk term can achieve $\sigma_{\boldsymbol{w}}^2 = 0$, thus $U(\boldsymbol{w}; \mathbf{R}) = \infty$. This means that (4.6) is be a safer bet, although with the added burden of assigning a level of risk aversion.

$$U(\boldsymbol{w}; \mathbf{R}) = \frac{\bar{r}_{\boldsymbol{w}}}{\sigma_{\boldsymbol{w}}^2} = \frac{\mathbf{1}^{\mathrm{T}} \mathbf{R} \boldsymbol{w}}{T \boldsymbol{w}^{\mathrm{T}} \boldsymbol{\Sigma} \boldsymbol{w}} \tag{4.7}$$

Finalization

When a portfolio has been through the rebalancing and is ready to be applied, a final stage might yet intervene. Finalization is a collective term for many last minute adjustments. Removal of negligible positions, stop loss and renormalization are important examples.

Small Positions Pruner. Portfolios might have peculiar miniature positions that are nothing but negligible. These small positions can be very expensive to constantly reapply, especially if a fixed cost is applied. It can be solved by simply requiring a minimum position by nullifying any position below a threshold followed by renormalization. This is a quick and dirty method, however, it is not meant to significantly alter the portfolio composition. It deals with marginal positions that can occur for technical reasons, like rounding errors.

Signal Flip. Sometimes the simplest methods, e.g., technical analysis, consistently underperform. In these cases it can be advantageous to simply reverse the signal the method gives. The meta-training phase determines if it is to be used case by case. In all honesty, this method was implemented as a intuitive response to extraordinarily bad returns generated by some technical analysis traders. The flip is performed by either flipping the portfolio positions or the price indicator movements, then applying renormalization. Given a portfolio or prediction $\boldsymbol{v} \in [0,1]^n$, flipping is performed by the transformation in (4.8) yielding a portfolio $\boldsymbol{w}$.

$$\boldsymbol{w} = \frac{\mathbf{1} - \boldsymbol{v}}{\mathbf{1}^{\mathrm{T}}(\mathbf{1} - \boldsymbol{v})} \tag{4.8}$$

Stop-Loss. Applying a stop-loss mechanism is a common method to try reducing losses. If the price of an asset drops below a certain threshold the mechanism automatically forces the position to be terminated. A version of stop-loss is the trailing stop-loss. It wipes out a position w_{ti} in $\boldsymbol{w}_t$ if it drops below a threshold $\gamma \in [0,1]$ relative to the maximum achieved price, i.e., $\max \boldsymbol{p}_i$. The lookback for the maximum achieved price is measured from the last rebalancing.

$$w_{ti} = \begin{cases} p_{ti} < \gamma \max \boldsymbol{p}_i & 0 \\ \text{else} & w_{ti} \end{cases} \tag{4.9}$$

Notice that wiping out a position requires the invested capital to be allocated to some other asset. This can be done either by renormalization or putting it in the risk-free asset.

4.2 Trading System Candidates

This section presents the concrete trading systems used tested in this project. 14 traders are formally tested in this project. More were built, but many did not either add any value or they performed too poorly in early experiments.

1. Autoregressive Trader (AR)
2. Multi-Layer Perceptron Neural Network Trader (MLPNN)
3. General Regression Neural Network Trader (GRNN)
4. Probabilistic Neural Network Trader (PNN)
5. Support Vector Machine Trader (SVM)
6. Support Vector Regression (SVR)
7. Multi-Horizon SVM Portfolio Selector Trader (MHS)
8. SVM-MLPNN Portfolio Selector (SM)
9. Efficient Frontier Trader (EF)
10. Rebalancing Efficient Frontier Trader (REF)
11. Extended Efficient Frontier Trader (EEF)
12. Relative Strength Index Trader (RSI)
13. Moving Average Convergence Divergence Trader (MACD)
14. Follow-the-Leaders Trader (FTL)

The different traders naturally form four categories. The simple technical analysis traders, machine learning traders applying price prediction, portfolio selector traders and efficient frontier traders. The former three apply adhere to the tenets of behavioral finance in that a market can have trends and heard mentality, while the later leans on modern portfolio theory with a dash of machine learning in some traders.

4.2.1 Technical Traders

The simplest traders apply technical analysis techniques directly. Many more techniques were initially tried out, though only a few showed any potential. These few are presented here and are tested at the same level as the other trading systems in the experiments.

Relative Strength Index Trader

The Relative Strength Index (RSI) Trader is based on the RSI indicator. This is essentially the ratio between the exponential moving averages of changes up and changes down. All parameters of the RSI are evolved and this makes it highly adjustable from evolution. As this trader has no other learning it depends heavily on meta-training to perform well.

Moving Average Convergence Divergence Trader

Moving Average Convergence Divergence Trader is another technical indicator. it uses nested moving averages of different lengths and generates a buy-signal based on these. The lengths of these moving averages are evolved making this highly dependent on meta-training as its only source of learning.

Follow-the-Leaders Trader

The Follow-the-Leaders Trader is quite frankly the simplest trading system of them all. It picks the N stocks with highest return form the T last observations. Both N and T are set by meta-training. In the nomenclature of technical analysis this could be called a momentum trading system as it is based on the assumption that rising stocks will continue to rise.

4.2.2 Machine Learning Traders

The machine learning traders are mostly variations on time series predictors that use various techniques to combine the predictions to a portfolio. All of these have the capability of evolutionary choosing which input data to use. For the data used one stream for each stock is given and the output for this is analysed to decide whether or not to buy the specific stock. How this is decided, and how much is bought varies among the traders.

All the traders in the category use learning. They are presented ideal data in much the same way. The major difference is whether they classify or do regression. Classifiers require some filtering of input data into predefined classes. For instance 0 for all negative values and 1 for all positive to predict price change direction.

Autoregressive Trader

The Autoregressive trader (AR) uses a autoregression node on each of a set of input data. The autoregression are given the actual time series changes as ideal and uses this to predict the next change. To create a portfolio of these predictions it chooses the n highest predicted changes and buys these. The amount bought is a normalized vector of the actual predictions, as such high predictions results in buying more. AR can also be set to buy a uniform portfolio and it can be filtered

to predict only price direction instead of magnitude. All these variations as well as the size of n are evolvable.

Multi-Layer Perceptron Neural Network Trader

The Multi-Layer Perceptron Neural Network Trader (MLPNN) uses MLPNNs to do time series prediction. As these handle multiple inputs and multiple outputs it only needs one network. As with AR it can do both regression and classification. Even though MLPNN could produce a portfolio directly this basic one only does predictions and uses the same technique as the AR trader to create portfolios. The MLPNN evolves the number of hidden layers, and the number of nodes in each and whether to be an Elman RNN or a standard feedforward network. This trader can either be run as a single standalone predictor or with filters on input and or output to facilitate different types of portfolio creation.

General Regression Neural Network Trader

The General Regression Neural Network (GRNN) Trader is similar to the MLPNN trader with a GRNN network to do the time series predictions. This trader has a lot fewer parameters that need setting, as topology and such is fixed, relative to the number of datapoints. When run on a large amount of datapoints this trader has poor performance due to the large network this requires. The GRNN is a regression predictor like AR and MLPNN.

Probabilistic Neural Network Trader

The Probabilistic Neural Network (PNN) Trader is the classification equivalent of the GRNN. This can only predict discrete classes of time series, rather than actual magnitudes. It is most commonly used to predict the direction of a time series. This is very useful for portfolio construction to contain stocks that are predicted to increase in price. The PNN trader can due to the classification aspect only be used for directional predictions, though initial experiments indicate that this is sufficient if not preferable.

Support Vector Machine Trader

The Support Vector Machine (SVM) Trader is yet another price predictor. The technical aspects are quite different from the ANNs, though the trader looks similar on a high level. The SVM trader only does classification like the PNN and the regression equivalent is the SVR trader. As mentioned earlier are SVM a very promising technique for price prediction. Several other trader use SVMs as a part to combine the time series prediction capability with other strategies. Through early experiments the SVM showed most promise as a time series predictor and were as such chosen as the de facto standard time series predictor.

Support Vector Regression

Support Vector Regression (SVR) Trader is the regression equivalent of the SVM Trader. The technical implementation is similar to the SVM. In the functional use it is more akin to AR, MLPNN and GRNN.

4.2.3 Portfolio Selector Traders

Multi-Horizon SVM Portfolio Selector Trader

The Multi-Horizon SVM Portfolio is a more complex trader than the other machine learning traders. A set of SVMs gives an prediction related to each stock. Each set of SVMs operate on different time horizons to achieve a broader perspective, and to identify patterns of different length. The predictions done by the SVMs are fed to an MLPNN that is used for the actual portfolio selection. This then chooses which prediction to use or several based on predictions done in a similar data context earlier. In effect this trader aim to choose among different or a combination of predictions from different horizons and to altogether increase the performance of the predictor. SVM is chosen as it is the best performing time series predictor.

SVM-MLPNN Portfolio Selector

The SVM-MLPNN Portfolio Selector (SM) trader s a variation on the MHS Trader describe above. Instead of only using a set of SVMs this uses one SVM and one MLPNN. All their parameters are evolved as in their respective normal traders and their predictions are again combined by another MLPNN. The reason for including MLPNN is due to its performance and ability to act risk averse. The SM trader and the MHS trader are both attempts at more intelligent portfolio selection than the simple uniform n approach.

4.2.4 Efficient Frontier Traders

According to modern portfolio theory the optimal portfolio choice is a mean-variance optimized portfolio adjusted the investors risk preference. The traders presented here are attempts to use efficient frontier portfolios to outperform the buy-and-hold benchmark of this project.

Efficient Frontier Trader

The Efficient Frontier (EF) Trader is a standard mean-variance optimized portfolio using quadratic optimization. The risk aversion factor determining the exchange ratio of variance to return is set by evolution. Aside from most of the other traders these are not aimed to exploit. These are aimed to find and hold the most efficient portfolio, independent of other actors in the market.

Rebalancing Efficient Frontier Trader

The Rebalancing Efficient Frontier (REF) Trader is the standard EF trader with a rebalancing control to limit the amount of trades to account for transaction costs. Experiments quickly showed the standard EF to trade to often to be feasible with transaction cost. The REF trader uses a rebalancing node that compares the historical performance of the suggested new portfolio versus the current and decides whether it is worth rebalancing depending on a threshold set by evolution.

Extended Efficient Frontier Trader

The Extended Efficient Frontier (EEF) Trader combines machine learning with efficient portfolio selection. The basis for portfolio created by this trader is a mean-variance optimized portfolio. A SVM is used for price prediction. These predictions are given an impact factor in which it can change the EF portfolio based on the predictions done. This factor is set by evolution. This trader is an attempt to include an exploitative aspect to the otherwise defensive EF portfolio. The EEF trader uses the rebalancing functionality from REF.

Chapter 5

Experiments and Results

This chapter goes into details about the performed experiments and their results. Section 5.1 covers the experimental setup, with limitations and assumptions. Section 5.2 goes into evaluation criteria that lay the ground work for evaluation of the trading systems performance. The trading systems experimental performance is measured quantitatively through statistical analysis of simulations conducted on historical data of two stock exchanges. Result on these datasets are presented in section 5.3.

5.1 Experimental Setup

In this section the various experiments which the trading systems have been put through are presented. The experiments in this section are built to isolate well performing trading system. The experiments are not only used to make a comparative evaluation of trading systems, but also used as a part of the search for optimal configurations.

5.1.1 Pre-experimental

In order to avoid an exhaustingly large experimental section some of the less tedious preliminary work has been left out. Before settling on a set of interesting traders a series of ad hoc tests were performed in order to establish a stable experimental foundation. The trading systems in question are not purely creative constructs. They have either shown promise through early informal experiments, or have some academic backing.

Meta-parameter Optimization

A central part of the framework is its meta-parameter optimization capabilities. Any parameter used in the system can be optimized either through grid search or evolution. This system was actively used in the preliminary phase to identify which parameters were relevant for each trading system to evolve. In the experimental run each system is given a meta-training phase where relevant parameters are evolved on a separate earlier dataset. The set of relevant meta-parameters varies from trading system to trading system. Multilayer perceptron neural networks, for instance, have meta-parameters that state how many nodes per layer and the number of layers to use. Note that the lower level training, such as tuning the weights between neurons, are separate from this meta-level optimization.

5.1.2 Experimental Environment

A central part of the experiments is the environment in which they take place. Obviously it is not feasible to run them on real markets. Lack of capital, reproducibility, integration and short time horizon are limitations that prevent applying the system to a real trading environment directly. These are limitations to the thesis scope, not the systems themselves. An important part of the framework is the market simulator. It is created for the purpose of testing and evaluating trading systems.

Market Simulation

Evaluation of trading systems requires an accurate simulation of the securities market it is applied to. This is a non-trivial task requiring simplifications that might introduce biases. The simulator uses real historical stock data, but many aspects of trading are hard to replicate. In real markets any actual trade will affect the market, if only to a small degree. In the simulator this effect is assumed to be negligible.

In real markets there are bid and ask price levels that defines what price other actors are willing to, respectively, buy and sell for. The difference between them is called the **spread**. The OBX dataset includes this information making it possible to use this information to create more realistic simulations. In the Dow dataset the adjusted close price must be used, due to this information being absent. A higher relative transaction cost is set to account for this limitation.

In real markets there might be limited supply or demand of a stock, this is ignored and transactions are assumed to always be executable. Since this will make trades cheaper than in real markets, the transaction cost is set higher than real market brokerage fees.

Typically, to perform trades one needs a broker to execute orders. These charge a fee or commission called a **brokerage cost**. In addition any costs incurred as an effect of an inefficient broker is called the **slippage**. Slippage and brokerage costs are generally much lower for automated traders as they can do most of the job itself. Most brokers also offer a fixed cost to handle an unlimited number of transactions. With a large enough initial investment this cost becomes negligible.

A number of actors in any marketplace aim to explicitly exploit other investors' transactions. This has increased lately with the advent of high-frequency traders. This effect can lead to a far worse price than the spread and price initially would indicate. As the trading systems operate on a daily basis they are not heavily exposed to these effects, yet a practical implementation needs to account for this problem.

Data Selection

There are two datasets used for these experiments.

1. Dow Jones Industrial Average (Dow) from 2000-2012
2. Oslo Stock Exchange 25 most liquid companies (OBX) from 2006-2012

Dow Jones Industrial Average. Dow Jones Industrial Average (Dow) is a well-known index comprising of major stocks from NYSE and NASDAQ. This often used and renowned index is well known and academically explored. For this reason it is a natural choice for experiments. The Dow is one of the world's most efficient and stable indices[51]. Bankruptcies and abnormal events are limited. Most changes are due to mergers and acquisition, and splits often due to antitrust litigations.

The Dow company composition is changed when required, based on a set of fixed rules. Dow's company composition count is small, consisting of only 30 stocks with large stable companies, leading to few changes. The experiments use the Dow composition at the start of the experiment and runs with this throughout the simulation. This way the bias of knowing whether some stock will go bankrupt or otherwise fail is eliminated. For instance General Motors filed for chapter 11 bankruptcy in 2011, an event present in the Dow dataset. Using the company composition from later than the simulation start date introduces a bias. This prunes out any stocks that have gone bankrupt or not performed well enough to remain in the index in the period.

The reason for not choosing a more numerous index is primarily hardware limitations and due to limited time to run simulations. The available computational resources limit how many companies can be analysed simultaneously. For practical

application, several hours will be available each day for computation and training. To be able to run simulations on more than 10 years' worth of data in a timely fashion requires a limited scope. This is done by requiring a limit on the number of companies and not performing relearning every single day on preliminary experiments.

One concession that was required for the Dow dataset was a special case in 2004 when SBC Communications bought AT&T and adopted its name and ticker, T. This led to issues obtaining original AT&T stock prices, forcing the simulations to be run without them. In the case of the General Motors bankruptcy, pre-bankruptcy data were salvaged and is present in the Dow dataset as Motors Liquidation, MTLQQ. Not a good stock to be exposed to.

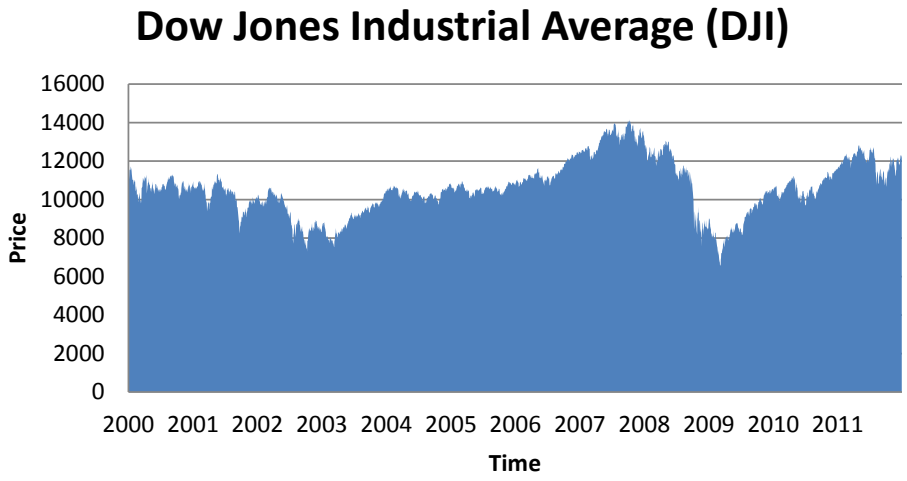


Figure 5.1: The Dow Jones Industrial Average index from 2000 to the end of 2011

OBX - Top Liquid Companies of Oslo Stock Exchange. OBX consists of the 25 most liquid companies from the Oslo Stock Exchange main index (OSEBX). This index composition is reviewed semi-annual. The experiments use data from the company composition which the index consisted of in February 2012, and their history back to 2006. Stocks with lacking history were removed, in total five companies. Since the OBX index companies are selected using the 2012 composition, a slight bankruptcy bias is introduced in that the companies are guaranteed not to go bankrupt in the simulated period. This is largely mitigated by evaluating the performance of the trading systems relative to the average index itself.

The experiments done on this dataset follow a set of stringent rules in order to be comparable to other studies using the same data. These rules will be discussed in more debt in the next section.

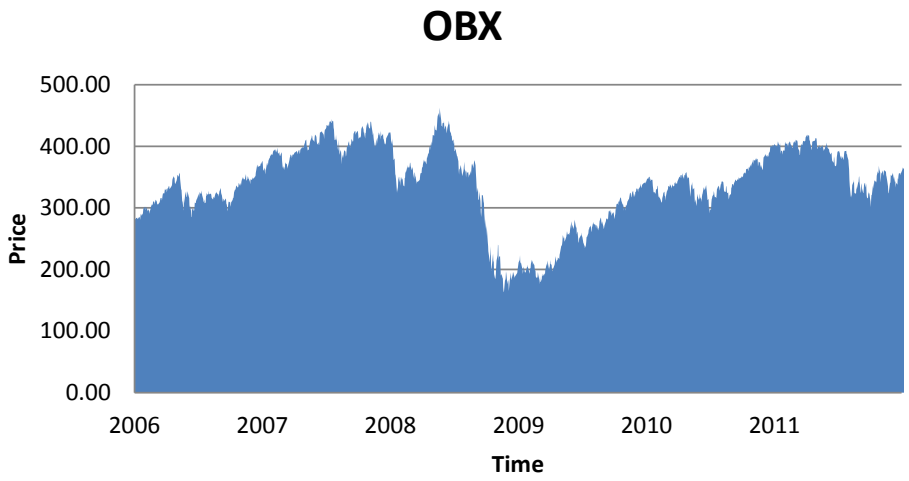


Figure 5.2: The OBX index from 2006 to the end of 2011

Dataset and Trading Rules

Every trading system mentioned in the trader algorithm section are run through a simulation with both the Dow and the OBX dataset. Both datasets have an initial part of the dataset reserved for meta-parameter optimization. The OBX experiments is intended to be comparable to other studies and follows a strict set of rules. In the Dow experiments the aim has been to simulate a real market as closely as possible with the data available. These rules are more closely fitted to work with this system. See table 5.1 for a comparison.

The main difference is the length of the dataset where Dow has 12 years and OBX 6 years. The OBX data contains information about bis and ask prices making a a real spread simulation possible. Note that this spread effect is present in all OBX experiments, with or without other transaction costs. The Dow set lack these features and uses a relative transaction cost to adjust appropriately for this. The transaction cost is generally considered to consist of the commission fee and the spread of the stock [96]. The transaction cost on Dow is relative to the transaction value and based on the average spread and commission fees. Dow Jones have one of the world's lowest spread at around 0.2% [31], this is used as transaction cost. By only trading after market close one achieves better prices, so this is a conservative estimate and should more than cover commission fees as well. The OBX data have this spread in them and due to the comparability with other simulations on these data there is a fixed transaction cost of 95 NOK per transaction.

Risk Free Asset. In order to simulate the option of putting a part of the portfolio into a risk-free-asset, there has been added a "safe" position to both simulation

Table 5.1: Overview of the experimental simulation setup. Notice how fixed transaction cost for OBX is set to 95/300,000. This is equivalent to a fixed cost of NOK 95 and an initial investment of NOK 300,000. Start case is set to just 1 for both datasets.

Simulation Set	Dow	OBX
Start Date	2000-01-03	2006-01-02
End Date	2011-12-31	2011-12-31
Fixed Transaction Cost	N/A	95/300000
Relative Transaction Cost	0.2%	N/A
Buy Price	Adj. Close	Ask Price
Sell Price	Adj. Close	Bid Price
Price Development	Adj. Close	Close
Meta-learning Dataset	3 years	1 year
Risk-free asset	US T-bills	Cash

runs. For the Dow dataset a 5 year US Treasury bond is used. It is computed by taking the N^{th} root of the daily reported annual yield, where N is equal to the average number of trading days in the year. This introduces a negligible variance. For the OBX simulations a constant cash option is used, with no return or variance.

Time Horizon and Data Resolution

An important question regarding the experiments are how long a time horizon the data should have, and what data resolution to use. For many trading systems questions about total run time, training set size, rebalancing and training interval and resolution must be considered.

The system runs over all available data. The first part is used for tuning and is not a part of the actual results. The rest is called simulation data. After an initial training period the system is ready to operate on the remaining data. The part the trading system is operating on is the simulated trading period. Throughout this period the system learns on an interval called the training interval. Some systems can adjust how often they produce a new portfolio; this is called the rebalance interval. The process is illustrated in figure 5.3.

The training set size will affect the length of patterns the system will be able to uncover. In order for a pattern to be learned, it needs patterns that reappear at regular intervals. With a short training set size, longer patterns will be hard to recognize. Another problem is that the system might think that the noise it is

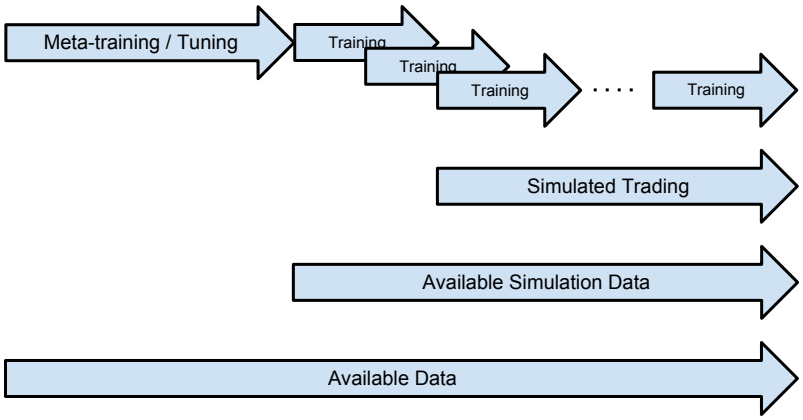


Figure 5.3: This figure illustrates a complete simulation run with the tuning phase. An initial meta-learning phase finds a locally optimal trader configuration, which is then trained in semi-online fashion.

observing is a pattern, i.e., overfitting. With too long dataset sizes, the trading system might learn trends which the market already has adjusted for, and lose reactivity.

The stock market movements shows multifractal properties[114] which means that roughly the same level of potential gains and losses can be achieved whatever resolution one choose to operate on. The main difference lies in that there has been shown a tendency for a risk premium, called the equity premium, in stock investments[106]. Stock markets tend to increase in value more than alternative markets, e.g., bonds, due to the risk premium for equity relative to safer investment alternatives. This makes long term investments a non-zero-sum game. With a short time horizon this premium is negligible, and investment can be thus be viewed as a close to a zero-sum game.

Most of the existing algorithmic traders present in the markets today are so-called high-frequency traders (HFT). As mentioned in section 2.1 these trade on a millisecond resolution. Most have naive strategies and achieve very low margins, but a high return due to the large amount of transactions. In this field there is very hard competition requiring the fastest hardware, network and software in order to achieve the highest profits. As argued earlier this is a zero-sum game, using principles that are not congruent with this system.

Many of a system's input features are not available at very high resolution. Trading systems should be able to hold positions over a longer time period, and benefit from the long term premium of stock markets. This is made possible by rebalancing depending on performance, and not on absolute time. Thus some of the systems

can dynamically adjust how long each position is held. One of the goals of the systems is to capture patterns of abnormal returns relative to risk and then react to exploit these. Systems need a low enough resolution to be reactive enough to exploit opportunities, as well as large enough to capture potential larger patterns, e.g. January effect.

Trading systems need a large datasets in order to avoid overfitting, however, Walczak found that a too large training sets degrades the performance of ANNs which promotes the use of a limited training set size [147].

The data used is at a daily resolution, much due to the availability. In the future works, section 6.5, experiments on lower resolution will be discussed. Daily resolution should give the system enough training data and avoid most of the competition from HFT. The length of the training set and the training interval are evolved. The training interval shows in most cases a sharp convergence toward the minimum, i.e. daily training.

5.1.3 Experimental Setup

This section presents the different experimental scenarios that are run, the aim with each, and how they are performed. There are two main sets of experiments, one for each of the mentioned datasets, Dow and OBX. For each set, every trading system is meta-parameter optimized on a subset of the data and then run and evaluated on the rest.

The Simulation Runs

All traders are simulate a live run over the trading data. This means that they submit a portfolio for each day and are only allowed to train on past data, thus no possibility for cheating through data leakage.

A subset of the traders is identified to be stochastic, i.e. producing different results with the same input and parameters. For instance, any trader with a neural network will be stochastic, as their neuron weights are initialized randomly. These traders are run for five simulations and the averages of those are used as the result.

The trading systems are evaluated on the average daily return they achieve relative to a benchmark, as well as on the variance relative to the benchmark. The benchmark is a uniform $1/N$ buy-and-hold portfolio strategy. Note that as different traders differ slightly in their training lengths the average benchmark they are compared with have different results as it is evaluated over differing time periods. They are still comparable through the relative performance against the benchmark.

All trading systems have meta-parameter optimization, while only some have their own low level learning. These have a trailing window of training data during the run, called the training set. The size of this is the aforementioned training set size, which is set as a meta-parameter.. Each time a system trains it starts from scratch with the new dataset and is thus only semi-online compared to one that had kept its state and only added the new datapoints. This way the classifier only trains on new and relevant data and does not need functionality for forgetting old datapoints gradually.

The system is evaluated on every portfolio created outside of the initial training period. A trading system with a training set size of 50 and a training interval of 1 will start producing portfolios from day 51. Then it will use the data from day 2 till 51 and train again to produce a portfolio for day 52 and so on. The trading system is evaluated from the portfolio at day 51.

5.1.4 Experimental Limitations

Stochasticity in Evolution

In the experiments evolution is used to find a set of suitable meta-parameters. Throughout the experiments the power of evolution as a dynamic trading system creator was discovered. To a large degree evolutionary optimization is responsible for the performance of the trading systems. This introduces another level of stochasticity in performance and reproducibility. To be fair, the systems were each given the exact same length of evolution to have the same basis.

Some of the trading systems still showed large variance in the outcomes from evolution. Most notable were the technical trading systems. The rationale for this is that evolution directly manipulates the inner workings of these simpler traders. For this reason the RSI and MACD were run 100 times each and the average of these runs used as results. Most others were not that volatile, and were only run once. In future work evolution should take a more automated and continuous role in training the traders. Traders and experiments will have to be built that account for its stochasticity.

5.2 Evaluation Criteria

This section is dedicated to the issue of evaluating the systems and this project as a whole. Evaluation is an issue much discussed in the field of artificial intelligence (AI). Cohen and Howe claims many papers do an inadequate job of evaluating the result of their AI systems[46].

The systems described in question in this thesis have an inherent black-box nature. This means it is inconceivable to understand the rationale behind why the system delivers its result. This complicates the evaluation. An evaluation with clear relative benchmarks is thus important and to avoid any pitfalls that invalidate the results. Examples of this are overfitting or data leakage. In this effort Cohen and Howe's five stage process is used for evaluating empirical AI systems [46]:

1. Find a task and a view of how to accomplish it
2. Refine the view to a specific method
3. Develop a program implementing the method
4. Design experiments to test the program
5. Run experiments

Ideally this development process should be run as iterative cycles. However, as the formal structure of this thesis and the previous project connected, the process has been divided into two phases. This project, the main thesis, largely contains the implementation, and naturally a bit of redesign. The identification of task and method were mostly completed in the previous project. Still the system created is made adaptable such that many different methods can be tested and thus a broader and more experiments can be run without the need to redesign the whole system.

The task is to beat the market index. In more quantifiable terms this entail to yield statistically significantly larger returns. In addition the goal is to do this consistently, thus the variance of these returns are of importance. The general hypothesis is that this can be achieve by the use of time series analysis and biologically inspired regression techniques. A set of specific systems is defined in the trading systems section 4.1. Instead of only creating a program to test one method, a framework has been made capable to test all these methods and to support further adjustments rapidly.

5.2.1 Evaluation

The system described in this paper is a practical work of engineering. The aim of this project is not to to evaluate some AI technique. It is designed with the goal to generate a good return on investment. Any evaluation of the AI techniques that the system employs is with that aim in mind. The system uses several complementary techniques in a competitive environment. This is meant to improve the system in its ultimate goal to generate a higher return than the rest of the market, not to test which technique is best. The main evaluation efforts will thus go to

measure different configurations of the system against standardized portfolios like indexes and conventional portfolio theoretical constructs. This paper is based on a hypothesis that neural network and evolution in a combination can be used to build effective portfolios, but there is no specific design in question. The experiments are thus more of an exploratory nature with the aim of identifying good portfolio building system with these techniques as central components. All the focus is on performance on a stock market and this is thus the only data sample the system will use.

Component Evaluation

The system consists of many different largely independent nodes that are used to build modules. As these modules are independent they can be tested as separate entities with much more ease than with a complete system test. The functionality of modules like data input, output and evaluation are common for all trading systems and tested for correctness independent of other components. These modules are deterministic and simple to test without need for further discussion. All components are individually tested before they are joined in larger more complex systems.

Portfolio Evaluation

The time series predictions can be tested on historical data, and the error easily calculated. The portfolio agents are harder to evaluate as there are no universally accepted optimal portfolio. The essential parameters are level of risk and return, the issue is how to evaluate the risk. A common approach is to use historical returns and variance to classify the quality of a portfolio, and then to compare this with an appropriate index. In this project a uniform buy-and-hold portfolio is used as an index or benchmark of which to compare results against. Specifically the portfolios are evaluated on:

- The mean excess daily return relative to the benchmark, $\bar{r}_p - \bar{r}_m$
- The variance relative to the benchmark, σ_p^2/σ_m^2
- The excess cumulative return compared with the benchmark, $R_p - R_m$

Here p denotes the portfolio, m the benchmark and R is the cumulative return over the period in question. As this project is a study of real-life application the excess cumulative return is weighed highest as this shows potential gains achievable. Over a longer time period the cumulative return will reflect both mean returns and variance.

System Evaluation

The eventual aim is to test the set of trading systems presented in section 4.1. The trading systems are in essence just portfolio producers and can be evaluated as described above. The trading systems are evaluated by the portfolios they hold throughout the experiments. The benchmark used to compare them to is a uniform buy-and-hold strategy. The reason for creating this $1/N$ benchmark instead of using the existing index to compare is that by creating this index with the system an area of potential bias is eliminated. This way any biases will also occur in the benchmark. Only differences relative to the benchmark are evaluated. This eliminates the market returns and risk and leaves only the difference the system has managed to produce.

5.3 Results

This section presents the actual results from the experiments described in the last section. The results is first objectively presented and then analyzed. Note that only the relative results to the benchmark are presented as the actual market returns are not relevant for this study.

5.3.1 Dow Jones Results

First the results from the Dow experiments are presented in a set of tables, later equivalent tables from the experiments on OBX follows. All experiments on the Dow set is run with a training period from 2000-2003 and the actual test from 2003-2012. Table 5.2 shows the results from test period without accounting for transaction costs. A set of portfolio measures to evaluate these results are presented in table 5.3. Table 5.4 shows the results from the experiments with transaction costs accounted for. Table 5.5 presents the evaluation metrics for these results. Note that the experiments with and without transaction costs are evolved separately. Different strategies are required depending on whether there are transaction costs. The results from the training periods of both these experiments are included in the appendix, section 7.3.

Tables explained

In these tables $\bar{r}_p - \bar{r}_m$ denotes the average excess return for the portfolio, p against the uniform buy-and-hold benchmark portfolio m. Any positive value is considered good here. σ_p^2/σ_m^2 is the ratio of the portfolio's variance versus the benchmark. Values in the range $[0, 1)$ are better than the benchmark. $max_p - max_m$ and $min_p - min_m$ shows the difference in the most extreme outliers. This provides a bit more information on the variation in the performance. A positive

Table 5.2: Results from test runs without transaction costs on DOW from 2003-01-01 to 2011-12-31

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
AR	3.262E-5	2.5901	-0.6591	0.1313	-0.1154
MLPNN	1.429E-5	0.9517	-0.2595	-1.232E-3	-2.347E-3
GRNN	-8.231E-6	1.0702	0.0439	2.183E-3	-2.591E-3
PNN	9.887E-5	0.9656	0.1254	-4.547E-3	6.512E-4
SVM	7.006E-6	1.3101	-0.3877	0.03	-0.0541
SVR	-1.497E-4	1.0062	-0.8598	6.356E-3	-0.0582
MHS	1.341E-5	0.9643	-0.1579	7.501E-3	-0.0194
SM	-8.638E-5	1.06	-0.5137	6.577E-3	-7.674E-3
EF	-9.172E-5	1.4835	-0.5604	-7.992E-3	-7.547E-3
REF	4.464E-4	4.4262	0.2988	0.3666	-0.1434
EEF	-1.104E-4	0.8838	-0.7344	-5.723E-3	3.142E-3
RSI	-2.025E-4	7.9025	-1.1905	0.3536	-0.2579
MACD	-3.327E-4	6.9779	-1.3693	0.3307	-0.2631
FTL	4.299E-5	0.9712	-0.084	-3.774E-3	-7.307E-5

difference is the best possible result, though these metrics are mostly for reference to the variance. $R_p - R_m$ is the excess cumulative return of the portfolio against the benchmark. Again, any positive value means the portfolio outperformed the benchmark. The reason for including both cumulative returns and average is that the average returns holds the most value in a scientific statistical context, while the cumulative is closer to the practical financial paradigm.

Table 5.5 and 5.3 presents a set of evaluation metrics for each trader. These include most of the common portfolio evaluation metrics presented earlier in section 2.1.4. Most of these account for risk and return and should together give a good foundation for comparison of the trading systems' performance. Note that some of these were not applicable if the trader committed to a buy-and-hold strategy or went all into the treasury bonds. In cases with traders the risk free asset omega will approach infinity and Sortino will divide by zero. In such cases these measures are presented as N/A for not applicable. Due to assumptions to use these metrics cannot be compared to external values of the same kind. The risk-free-rate is taken from the annual yield of the treasury bond included among the assets. The scope of these metrics is daily, that means that for instance Jensen's alpha is comparable

Table 5.3: Evaluation of trading systems from experiments without transaction cost on DOW from 2003-2012

Trader	α	β	Sharpe	Treynor	Sortino	Ω
AR	1.582E-4	0.4136	0.0112	5.968E-4	0.0227	1.1198
MLPNN	1.96E-5	0.9788	0.0198	2.712E-4	0.0337	1.5268
GRNN	-1.584E-5	0.9681	-0.0164	-2.55E-4	-7.129E-3	0.9595
PNN	9.868E-5	1.0012	0.0193	2.592E-4	0.0353	1.2367
SVM	4.799E-5	0.805	0.0139	2.698E-4	0.0275	1.157
SVR	-3.816E-5	0.5505	7.146E-3	1.787E-4	0.0193	1.1669
MHS	2.175E-5	0.9486	0.013	1.847E-4	0.0275	1.1863
SM	-7.325E-5	0.9148	4.808E-3	7.362E-5	0.0171	1.1437
EF	-1.996E-5	0.4875	2.859E-3	9.91E-5	0.0133	1.2056
REF	6.169E-4	0.3398	0.0244	2.074E-3	0.038	1.133
EEF	-1.049E-4	0.9769	9.683E-3	1.279E-4	0.0244	1.1318
RSI	-4.798E-5	0.2131	2.431E-3	-7.895E-4	6.818E-3	1.4435
MACD	-1.957E-4	0.302	-1.349E-3	-1.185E-3	3.296E-3	2.3553
FTL	4.098E-5	1.0102	0.0177	2.369E-4	0.0339	1.2157

to daily returns.

Analysis

Here the results are shortly analyzed and the findings summarized. A more complete analysis of different aspects follows in the discussion, in section 6.2.

Without Transaction Costs. Few traders manage to outperform the benchmark even without transaction costs in the Dow experiment. A good deal have slightly higher average daily returns, but coupled with high variance this leads to poor cumulative returns in most cases. REF, GRNN and PNN achieves higher cumulative returns than the benchmark. The GRNN is basically equal to the benchmark. REF has a dangerously high variance explaining why the cumulative returns do not fully correspond to the great average daily returns. PNN with the second highest cumulative return 12.5 pp over benchmark actually dominated the benchmark, though barely. In terms of Sharpe and Sortino ratios it is still beaten by REF. Summed up the ANNs based and EF based traders performs best in this experiment, though only 3 of 14 trades could be said to perform better or equal compared to the benchmark. It is worth noting that the technical predictors, RSI

Table 5.4: Results from test runs with transaction costs on DOW from 2003-2012

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
AR	-6.103E-4	5.4127	-1.7785	0.2504	-0.4371
MLPNN	-7.497E-5	0.9642	-0.4916	-3.689E-3	-1.026E-3
GRNN	5.353E-6	1.078	-0.3153	4.089E-3	-2.921E-3
PNN	-3.738E-4	0.6002	-1.0895	-0.0221	-2.274E-3
SVM	-7.259E-5	2.5068	-1.0224	0.0869	-0.1242
SVR	-3.121E-4	0.7664	-0.9881	6.356E-3	-0.0153
MHS	-4.78E-4	4.4448	-1.3755	0.4288	-0.3487
SM	-1.37E-4	1.8472	-0.7398	0.1111	-0.0572
EF	-2.17E-4	2.0316	-1.1129	0.0132	-0.0625
REF	-1.199E-4	2.6552	-1.237	0.1147	-0.0787
EEF	-2.295E-4	1.667	-0.8964	0.0604	-0.0603
RSI	-6.47E-4	11.059	-1.6601	0.4373	-0.3225
MACD	-4.813E-4	8.3107	-1.659	0.4019	-0.2982
FTL	-8.022E-5	0.9713	-0.5284	-3.912E-3	-2.11E-4

and MACD, perform truly miserable.

Results With Transaction Costs. With transaction costs no trader is able to challenge the benchmark. The two best performers both fare badly this time around. REF is dragged down by the enormous transaction costs it inflicts. Figure 5.4a shows a snapshot of its portfolio composition during a run, for comparison the portfolio composition of GRNN in figure 5.4b. The GRNN composition here is essentially the same as the benchmark holds. The optimal solution is assumed to be somewhere in between these two extremes.

PNN is evolved to become a quite different and more defensive trader and ends up with impressively low variance, though with very poor returns. The best performers are the GRNN with only 31 pp cumulative return less than the benchmark, and this holds largely the same portfolio as the benchmark with the addition of treasury bonds which puts the return behind the benchmark.

Table 5.5: Evaluation of trading systems from test runs on DOW from 2003-2012

Trader	α	β	Sharpe	Treynor	Sortino	Ω
AR	-4.534E-4	0.2218	-0.0128	-1.843E-3	-0.0103	1.1401
MLPNN	-7.061E-5	0.9758	7.794E-3	1.086E-4	0.0195	1.5011
GRNN	1.251E-5	0.9635	0.0142	2.093E-4	0.0285	1.2116
PNN	-3.942E-4	1.1141	-0.0184	-1.752E-4	-8.193E-3	1.4227
SVM	4.171E-5	0.5283	7.819E-3	3.213E-4	0.0167	1.1102
SVR	-2.429E-4	0.6359	-0.0102	-1.918E-4	5.976E-5	1.3036
MHS	-3.631E-4	0.3059	-0.0104	-1.419E-3	-7.214E-3	1.0724
SM	-7.267E-5	0.5886	1.697E-3	9.658E-6	0.0111	1.0879
EF	-9.114E-5	0.3755	-7.918E-4	-4.114E-5	7.06E-3	1.1117
REF	-4.16E-6	0.5516	6.181E-3	2.506E-4	0.014	1.131
EEF	-1.701E-4	0.6237	-4.04E-3	-1.148E-4	3.919E-3	1.0554
RSI	-4.842E-4	0.1709	-8.644E-3	-3.777E-3	-6.944E-3	1.533
MACD	1.007E-4	0.2488	4.812E-3	0.554	3.785E-3	1.2698
FTL	-8.223E-5	1.0102	8.585E-3	1.15E-4	0.0224	1.1898

5.3.2 OBX Results

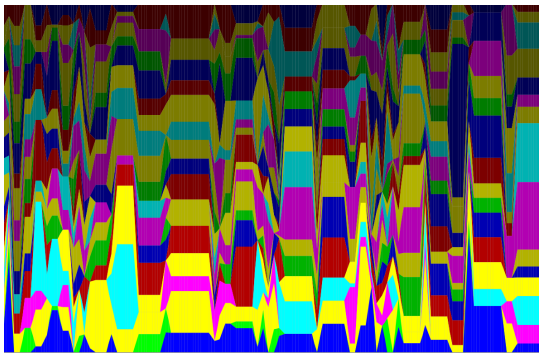
Analysis

Here the results are shortly analyzed and the findings summarized. A more complete analysis of different aspects follows in the discussion, in section 6.2.

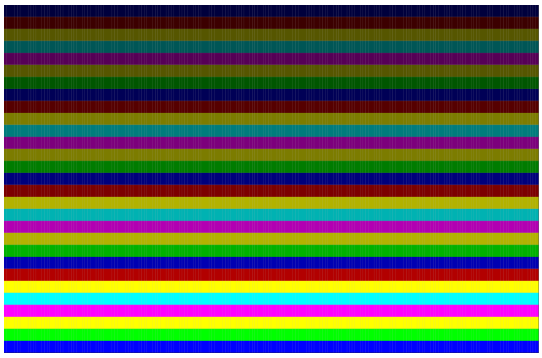
Results Without Transaction Costs. In the OBX experiments without transaction costs most trading systems delivers cumulative returns above the benchmark. The SVR trader has the highest cumulative return though it is surpassed by both EEF and SVM in mean returns. These two have a much higher variance, which explains why the SVR end up with the highest cumulative return. Due to this low variance the SVR also dominate most risk-adjusted measures.

The technical indicators, RSI and MACD, does surprisingly well in this experiment. RSI has higher return while MACD has a lower variance, but in terms of Sharpe ratio they perform equal.

The ANNs performs more or less alike. The GRNN is slightly better, though all are relatively close to the benchmark. MHS and SM require more training data than the other, and seem not to have gotten enough for any adequate solution in



(a) This is a snapshot of the portfolio composition of the REF trader in its test run on Dow without transaction costs. Days are at the x-axis with distribution on the y-axis. This illustrates the trading pattern it uses. The large number of trades demonstrates why this approaches fails with the advent of transaction costs.



(b) This is a snapshot the portfolio composition the GRNN trader holds during the test run on Dow without transaction costs. Days are at the x-axis with distribution on the y-axis. A snapshot of the portfolio composition for the SM trading system when running with transaction cost. The position is, as illustrated, constant.

this sample. EF performed quite well, while the REF was outperformed by the benchmark. FTL and AR are below the benchmark.

Results With Transaction Costs. Few of the traders in this experiment are much affected by the transaction costs. The most notable exceptions are the support vector traders and FTL. Some, like EF and MACD seems actually to be positively affected. This can be randomness, or the fact that evolution, with the added pressure of transaction costs, creates a more general solution. Another possibility is that the fixed transaction costs are less destructive than the relative ones used in the Dow experiments. Note that the spread effect is present in both the OBX experiments, further limiting the difference.

EEF achieves the highest return, and dominates most of the risk-adjusted measures. EF is not far behind; it has far less return, but also far less variance. The technical traders, RSI and MACD, still perform well, with similar performance, though less return than EEF and higher variance than EF.

Among the ANNs the GRNN's performance has degraded further than the others' with the added transaction costs. This leaves PNN as the highest cumulative return among the ANNs. The PNN uses a purely buy-and-hold strategy and is thus unaffected by transaction costs.

The volatility of the SVR has increased with transaction costs, resulting in a far

Table 5.6: Results from test runs without transaction costs on OBX from 2007-2012

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
AR	-4.617E-4	1.2893	-0.3036	0.017	-0.0147
MLPNN	-2.751E-5	1.0451	0.1711	-1.922E-3	-5.721E-3
GRNN	1.814E-5	1.1025	0.2101	5.6E-3	-5.422E-3
PNN	-1.471E-5	0.9456	0.208	3.557E-5	5.079E-3
SVM	9.658E-4	2.7268	1.1001	0.0611	-0.0327
SVR	6.215E-4	0.7598	1.4045	5.411E-3	-0.0135
MHS	-7.134E-4	0.9816	-0.4195	4.32E-4	-6.11E-3
SM	-6.255E-4	1.2189	-0.3827	0.0132	-0.0316
EF	-1.257E-5	0.7515	0.2824	-4.46E-3	0.0188
REF	-8.409E-5	1.3687	0.0246	0.0277	-0.0369
EEF	1.093E-3	3.0148	1.0816	0.0803	-0.2069
RSI	4.843E-4	2.5195	0.7845	0.0936	-0.0902
MACD	1.675E-4	1.0142	0.4947	9.112E-3	1.221E-3
FTL	-2.392E-4	0.9844	-0.1112	2.804E-3	-2.239E-3

worse cumulative return. The SVMs return has degraded with the introduction of transaction costs though the variance stays stable.

SMs performance increases with the introduction of transaction and it ends up closely resembling the benchmark. The MHS, REF and AR still underperforms severely, thought the AR has managed the lowest variance in the experiment.

Table 5.7: Evaluation of trading systems from experiments without transaction cost on OBX from 2007-2012

Trader	α	β	Sharpe	Treynor	Sortino	Ω
AR	-3.965E-4	0.7931	-5.698E-3	-1.848E-4	-7.81E-3	1.0394
MLPNN	-1.781E-5	0.9717	0.0135	3.263E-4	0.0178	1.1549
GRNN	3.51E-5	0.9533	0.0157	3.997E-4	0.0206	1.1828
PNN	-2.209E-5	1.022	0.0148	3.14E-4	0.0194	1.1558
SVM	1.189E-3	0.4063	0.0357	3.303E-3	0.0544	1.0601
SVR	6.801E-4	0.83	0.0481	1.164E-3	0.0742	1.1475
MHS	-7.012E-4	0.968	-0.0145	-3.405E-4	-0.019	1.0501
SM	-5.595E-4	0.7962	-0.0121	-3.789E-4	-0.0166	1.009
EF	1.322E-4	0.5953	0.0174	5.799E-4	0.0234	1.1315
REF	-1.47E-5	0.8054	0.0101	3.384E-4	0.0134	1.1205
EEF	1.291E-3	0.4237	0.0359	3.393E-3	0.0492	1.0683
RSI	7.133E-4	0.4252	0.0259	2.236E-3	0.0378	1.0916
MACD	2.009E-4	0.916	0.0259	6.177E-4	0.0351	1.1056
FTL	-2.411E-4	1.0049	7.398E-3	1.584E-4	9.616E-3	1.1402

Table 5.8: Results from test runs with transaction costs on OBX from 2007-2012

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
AR	-6.183E-4	0.5401	-0.2697	0.0172	-0.034
MLPNN	-1.762E-5	1.0015	0.1945	-1.087E-3	-2.861E-3
GRNN	-1.261E-6	1.311	0.0979	0.0213	-0.0168
PNN	-1.62E-5	0.9457	0.2026	3.557E-5	5.079E-3
SVM	1.088E-4	2.5627	-0.1024	0.0952	-0.0835
SVR	3.639E-4	4.2644	-0.2649	0.1529	-0.3266
MHS	-5.229E-3	1.6792	-0.8889	-0.069	-0.5511
SM	-1.578E-4	1.0022	0.0162	7.783E-3	-4.227E-3
EF	2.619E-4	1.0016	0.6533	0.0779	0.0175
REF	-5.299E-3	2.8108	-0.8797	0.0326	-0.7088
EEF	1.074E-3	3.0133	1.0423	0.0803	-0.2069
RSI	3.837E-4	2.3963	0.7011	0.0851	-0.0971
MACD	2.546E-4	2.1991	0.5258	0.0728	-0.0741
FTL	-3.138E-3	1.0007	-0.9992	2.884E-4	-3.594E-3

Table 5.9: Evaluation of trading systems from experiments with transaction costs runs on OBX from 2007-2012

Trader	α	β	Sharpe	Treynor	Sortino	Ω
AR	-4.28E-4	0.4224	-0.0174	-6.838E-4	-0.0124	NA
MLPNN	-1.507E-5	0.9929	0.0149	3.46E-4	0.0196	1.15
GRNN	5.296E-5	0.8392	0.0128	4.003E-4	0.0172	1.091
PNN	-2.338E-5	1.022	0.0142	3.042E-4	0.0187	1.1621
SVM	3.122E-4	0.3793	0.0121	1.151E-3	0.018	1.0156
SVR	6.172E-4	0.309	0.0153	2.364E-3	0.0202	1.0228
MHS	-4.931E-3	0.1179	-0.1642	-0.0415	-0.0729	N/A
SM	-1.509E-4	0.9821	9.867E-3	2.302E-4	0.0131	1.1404
EF	4.218E-4	0.5525	0.0283	1.121E-3	0.0427	1.1733
REF	-5.023E-3	0.2259	-0.1279	-0.0219	-0.0685	N/A
EEF	1.272E-3	0.4238	0.0354	3.346E-3	0.0486	1.0683
RSI	6.144E-4	0.421	0.0216	2.062E-3	0.0318	1.2737
MACD	4.406E-4	0.533	0.02	1.581E-3	0.0294	1.0928
FTL	-3.131E-3	0.9809	-0.1263	-2.793E-3	-0.1664	0.8581

Chapter 6

Summary and Contributions

6.1 Summary

This thesis creates a series of trading systems using machine learning techniques and evolutionary algorithms. These are tested on a dataset from Dow Jones Industrial Average (Dow) and the OBX index from Oslo Stock Exchange. The Dow dataset was found to be efficient and with little possibility of generating excess returns on. In the OBX data some degree of inefficiency were found. High excess returns can be achieved, though at high risk. The best returns at the same risk are achieved by a trader using efficient portfolios. The conclusion is that no trading system was found that can achieve excess profits beyond what any index fund can achieve. The results from this study support the efficient market hypothesis.

Support vector machines and support vector regression showed the most promise as time series predictors in the trading systems. They do however have to be constrained as not to trade too often and incur heavy transaction costs. Artificial Neural Networks were found to give somewhat good results, though they behaved mostly like buy-and-hold strategies.

Evolution was used to tune the meta-parameters of the traders. It was found to be very powerful and future works will include evolution as an integral part of the actual trading system. Evolution was found to be able to incorporate risk indirectly when only given return as a utility function, though over a longer period of time.

6.2 Discussion

The discussion reviews more qualitative aspects of the results. First, the trading systems are individually analyzed. Then the goals are revisited. The rest is dedicated to other important concepts like market efficiency and risk.

6.2.1 Trading System Analysis

In this section the results for the individual trading systems are more thoroughly analyzed in context of the experiments on both datasets, with and without transaction costs.

AR

Table 6.1: AR results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	3.262E-5	2.5901	-0.6591	0.1313	-0.1154
Dow w/TC[1]	-6.103E-4	5.4127	-1.7785	0.2504	-0.4371
OBX	-4.617E-4	1.2893	-0.3036	0.017	-0.0147
OBX w/TC	-6.183E-4	0.5401	-0.2697	0.0172	-0.034

The autoregression trader is included in this project mostly as an academic exercise, in addition to being a benchmark for machine learning techniques. In all experiments this trader was among the worst performers, see summarized results in table 6.1. The best result it receives is with transaction costs on OBX, where it ends up with only 54% of the benchmark variance, though with quite a drop in return. The fact that AR does not outperform the benchmark even without transaction cost leads to the conclusion that this trader has no significant prediction capability.

MLPNN

Table 6.2: MLPNN results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	1.429E-5	0.9517	-0.2595	-1.232E-3	-2.347E-3
Dow w/TC	-7.497E-5	0.9642	-0.4916	-3.689E-3	-1.026E-3
OBX	-2.751E-5	1.0451	0.1711	-1.922E-3	-5.721E-3
OBX w/TC	-1.762E-5	1.0015	0.1945	-1.087E-3	-2.861E-3

In the Dow experiments the MLPNN performs very similar to the benchmark. It does move go all-in into treasury bonds at times. This degrades the return, especially when transaction costs are applied. It is outperformed by GRNN both with and without transaction costs. It is worth noting that the MLPNN aside from GRNN actually do some switching between different portfolios, where GRNN buys and holds all stocks throughout the run. Still both are pretty static and show little predictive power.

The performance is similar in the OBX case, though it manages to outperform the benchmark in both cases. Relative to the other traders the MLPNN has low returns, though it has among the best variance ratios. In the OBX experiments the MLPNN rebalances seldom and is not really affected by transaction costs. The improvement with transaction costs is too small to be significant. In general it holds a uniform portfolio, but a subset of the benchmark with only 9-10 stocks.

All in all the MLPNN is stable and maintains a low variance. It does not seem to be able exploit opportunities to the same degree as other traders like the EEF or the SVM/R.

GRNN

Table 6.3: GRNN results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	-8.231E-6	1.0702	0.0439	2.183E-3	-2.591E-3
Dow w/TC	5.353E-6	1.078	-0.3153	4.089E-3	-2.921E-3
OBX	1.814E-5	1.1025	0.2101	5.6E-3	-5.422E-3
OBX w/TC	-1.261E-6	1.311	0.0979	0.0213	-0.0168

GRNN performs well in the Dow experiment relative to the other traders. This is not due to any predictability, rather than through evolution the GRNN has evolved to one stable strategy. It buys a uniform portfolio of all stocks; this means that the GRNN is tweaked to always recommend all stocks. The reason for the slight difference is that GRNN also has treasury bonds available and these are a part of the portfolio. Relative to the other traders' attempts, this is a good choice; however this choice must be attributed to evolution and not the GRNN trader itself. In the training period the GRNN has not managed to outperform the benchmark, and the best solution evolution finds is the ones were the GRNN always recommend a type of uniform buy-and-hold strategy. This is an interesting result, but does not give any value to the GRNN trader.

In the OBX experiments it performs on par with the rest of the ANNs with some

excess return above the benchmark. Its performance is degraded somewhat by transaction costs and both return measures degrade and variance increase.

In summary GRNN is very defensive and holds a portfolio closely resembling the benchmark at most times. The main reason for its excess cumulative return at OBX is that the benchmark underperforms the risk-free-rate, and GRNN has some position in this rate. The GRNN is rather static and does not adaptability in itself, any changes in behavior is brought by evolution. The conclusion drawn from this is that GRNN in itself have little or no predictive ability.

PNN

Table 6.4: PNN results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	9.887E-5	0.9656	0.1254	-4.547E-3	6.512E-4
Dow w/TC	-3.738E-4	0.6002	-1.0895	-0.0221	-2.274E-3
OBX	-1.471E-5	0.9456	0.208	3.557E-5	5.079E-3
OBX w/TC	-1.62E-5	0.9457	0.2026	3.557E-5	5.079E-3

PNN is among the best traders in the Dow experiments. It is the only trader to dominate the benchmark without transaction costs. It is not far superior to the benchmark, but has slightly lower variance and 12.5 pp higher cumulative returns. The returns dwindle sharply with the introduction of transaction costs. At the same time the variance drops to only 60% of the benchmark. The loss in return is still too severe to be mitigated by the low variance.

In the OBX experiments the PNN uses largely a buy-and-hold strategy with little rebalancing and is thus not much affected by transaction costs. It achieves positive cumulative return with a bit less variance than the benchmark; the mean excess return is close to zero. The PNN is slightly better than the benchmark in the OBX case, and worse in the Dow case. All in all it is no safe bet against the benchmark and far from significantly better.

SVM

In the Dow experiments SVM rebalances often and still underperforms relative to the benchmark on both cumulative return and variance. The frequent trades are especially damaging for performance with transaction costs, and in this case SVM is severely outperformed by the benchmark. In the OBX experiments the SVM is still rebalancing frequently. The variance is far above the benchmark. Introduction of transaction costs severely degrades the returns. The negative excess cumulative

Table 6.5: SVM results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	7.006E-6	1.3101	-0.3877	0.03	-0.0541
Dow w/TC	-7.259E-5	2.5068	-1.0224	0.0869	-0.1242
OBX	9.658E-4	2.7268	1.1001	0.0611	-0.0327
OBX w/TC	1.088E-4	2.5627	-0.1024	0.0952	-0.0835

return in the transaction cost case is attributable to the large amount of trades made. The ANNs all manage positive excess cumulative returns with far less mean excess returns.

The SVMs performance when unconstrained by transaction costs is good and indicates that it may in fact possess predictive ability. What is clear is that to blindly follow these prediction results is a very risky endeavor, as well as incurring high transaction costs. SVM might be good in combination with other methods, but alone it cannot outperform the benchmark.

SVR

Table 6.6: SVR results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	-1.497E-4	1.0062	-0.8598	6.356E-3	-0.0582
Dow w/TC	-3.121E-4	0.7664	-0.9881	6.356E-3	-0.0153
OBX	6.215E-4	0.7598	1.4045	5.411E-3	-0.0135
OBX w/TC	3.639E-4	4.2644	-0.2649	0.1529	-0.3266

In the Dow experiment the SVR delivers consistently poor results, although with a low variance. In the OBX experiments the SVR performs very well without transaction costs. It is the best performer in that experiment. The introduction of transaction costs degrades the performance of the SVR dramatically. The variance skyrockets and the return falls, resulting in a far worse cumulative return, below the benchmark.

MHS

Similar to the SVM the MHS is strongly affected by transaction costs. These increase variance and decrease returns. Without transaction costs MHS is one of few traders that performs worse on OBX compared to the Dow experiments. The rationale for this might be that the MHS trader requires more than double the

Table 6.7: MHS results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	1.341E-5	0.9643	-0.1579	7.501E-3	-0.0194
Dow w/TC	-4.78E-4	4.4448	-1.3755	0.4288	-0.3487
OBX	-7.134E-4	0.9816	-0.4195	4.32E-4	-6.11E-3
OBX w/TC	-5.229E-3	1.6792	-0.8889	-0.069	-0.5511

amount of data that other traders do. This leaves too little training data to achieve good generalization, which can explain the poor performance. MHS improved on the pure SVM's performance in only one case and it is altogether no improvement of the SVM trader.

SM

Table 6.8: SM results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	-8.638E-5	1.06	-0.5137	6.577E-3	-7.674E-3
Dow w/TC	-1.37E-4	1.8472	-0.7398	0.1111	-0.0572
OBX	-6.255E-4	1.2189	-0.3827	0.0132	-0.0316
OBX w/TC	-1.578E-4	1.0022	0.0162	7.783E-3	-4.227E-3

The SM is an attempt to combine the risk averse behavior of the MLPNN with the predictability of the SVM. The return is evened out and it is less reactive to transaction costs than SVM. The returns are however still low. It performs well with transaction costs on OBX. An unfortunate result is that MLPNN stochastically dominates SM, doing nothing to improve results.

EF

Table 6.9: EF results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	-9.172E-5	1.4835	-0.5604	-7.992E-3	-7.547E-3
Dow w/TC	-2.17E-4	2.0316	-1.1129	0.0132	-0.0625
OBX	-1.257E-5	0.7515	0.2824	-4.46E-3	0.0188
OBX w/TC	2.619E-4	1.0016	0.6533	0.0779	0.0175

EF does not perform well in the Dow experiment. It has low returns and a high variance. The poor result with transaction costs can be explained by frequency of rebalancing EF does. Why EF underperforms without transaction costs is harder to explain. It has both high variance and low returns which makes little sense for an efficient frontier trader. The high variance indicates low risk aversion which the parameters confirm. The likely explanation is that it was evolved to be risky during the training period, and then was punished for this in the test period with events like the financial crisis.

In the OBX experiments EF is the arguably the best performer. It does not have quite the return of EEF. It does however still outperform the benchmark and with no higher variance, where EEF got 3 times the benchmark variance.

REF

Table 6.10: REF results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	4.464E-4	4.4262	0.2988	0.3666	-0.1434
Dow w/TC	-1.199E-4	2.6552	-1.237	0.1147	-0.0787
OBX	-8.409E-5	1.3687	0.0246	0.0277	-0.0369
OBX w/TC	-5.299E-3	2.8108	-0.8797	0.0326	-0.7088

REF was created to reduce the number of trades which EF executes. The goal was to limit the impact of the transaction costs. This clearly failed. On every single result except on Dow without transaction costs the REF is dominated by EF. It achieves better return on Dow with no transaction costs, but at a far higher variance. As to why this approach failed: One hypothesis is that with all the extra parameters resulting from adding rebalancing, REF was able to overfit to the training set. This makes REF performance poorly on the test set. In any case, this is clearly not an improvement on either the plain EF or the benchmark.

EEF

The EEF is another extension of EF. It uses the regular EF in a combination with an SVM and a rebalancer. If variance is accounted for EEF performs equal or better than EF on the Dow experiments. In the OBX experiments the EEF trader is the best when using transaction costs and among the best without. In conclusion, though not performing very well on Dow, EEF is one of the better traders in these experiments.

Table 6.11: EEF results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	-1.104E-4	0.8838	-0.7344	-5.723E-3	3.142E-3
Dow w/TC	-2.295E-4	1.667	-0.8964	0.0604	-0.0603
OBX	1.093E-3	3.0148	1.0816	0.0803	-0.2069
OBX w/TC	1.074E-3	3.0133	1.0423	0.0803	-0.2069

Table 6.12: RSI results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	-2.025E-4	7.9025	-1.1905	0.3536	-0.2579
Dow w/TC	-6.47E-4	11.059	-1.6601	0.4373	-0.3225
OBX	4.843E-4	2.5195	0.7845	0.0936	-0.0902
OBX w/TC	3.837E-4	2.3963	0.7011	0.0851	-0.0971

RSI

The RSI is a very simple trader. Every aspect of its behavior is set by evolution, making it is very dependent on the training it receives. It performs terribly on the Dow dataset, flunking on all measure available. After the introduction of transaction costs the RSI nearly goes bankrupt. Note that this is an extraordinary feat in the Dow experiments as all transaction costs are relative. These results are in remarkable contrast with the OBX results. Here RSI gives strong and stable results. This indicates that the trading strategy which was evolved during training is applicable through most of the period, in Dow the solution did not work outside of training. This could be due to the longer test period of the Dow dataset; however test were done with a smaller period much like the OBX experiment and the RSI underperformed to the same extent. This effect could be attributable to the fact that inefficiencies persist to a larger degree in the smaller and less efficient OBX index, than on Dow[18].

MACD

MACD is conceptually similar to the RSI as it too is a technical indicator solely dependent on evolution for its performance. MACD has less dispersion in performance. It is still among the worst performers on the Dow experiments and delivers good results on the OBX experiments. This result further strengthens the evidence that OBX is less efficient that Dow as with the RSI result.

Table 6.13: MACD results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	-3.327E-4	6.9779	-1.3693	0.3307	-0.2631
Dow w/TC	-4.813E-4	8.3107	-1.659	0.4019	-0.2982
OBX	1.675E-4	1.0142	0.4947	9.112E-3	1.221E-3
OBX w/TC	2.546E-4	2.1991	0.5258	0.0728	-0.0741

FTL

Table 6.14: FTL results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow	4.299E-5	0.9712	-0.084	-3.774E-3	-7.307E-5
Dow w/TC	-8.022E-5	0.9713	-0.5284	-3.912E-3	-2.11E-4
OBX	-2.392E-4	0.9844	-0.1112	2.804E-3	-2.239E-3
OBX w/TC	-3.138E-3	1.0007	-0.9992	2.884E-4	-3.594E-3

FTL is the simplest trader and introduced as a worst case benchmark and to evaluate whether the systems were able to cheat. It has, not surprisingly, very poor performance, in its best case it closely mimics the benchmark by picking the $n-1$ winners in an n stock index. This approach is enough to be among the best performers on Dow with transaction cost.

Summary

In the OBX experiments the Extended Efficient Frontier trader is a clear winner in terms of return. The pure EF have less return, but no higher variance than the benchmark, while EEF got three times higher. The difference in performance among the simpler technical trader, RSI and MACD, indicates that OBX might be less efficient than the Dow. This would further explain the why most traders performed far better on OBX than Dow, this point is discussed further in section 6.2. All in all no trader performs well on Dow, while the efficient frontier based ones are best on OBX.

Scenario Analysis

In this section the EEF trader is run through specific historical events to evaluate its performance and adaptability. It is mapped against the benchmark for comparison. This is not meant as a rigorous statistical experiment, but as a demonstration run for the best trader. It is interesting to see how EEF handles different scenarios

as the financial crisis and is intended to identify qualitative strengths and weaknesses. Note that all these runs are done with transaction costs. The scenarios are summed up in table 6.15.

Table 6.15: The Scenarios to be run in the analysis and a short description

Period	Time	Description
The Dow test period	2003-2012	The period used for testing in the Dow experiments
The OBX test period	2007-2012	The period used for testing in the OBX experiments
The Financial Crisis on Dow	2008-2009	A period marked by large turmoil in the market.
The Financial Crisis on OBX	2008-2009	A period marked by large turmoil in the market.

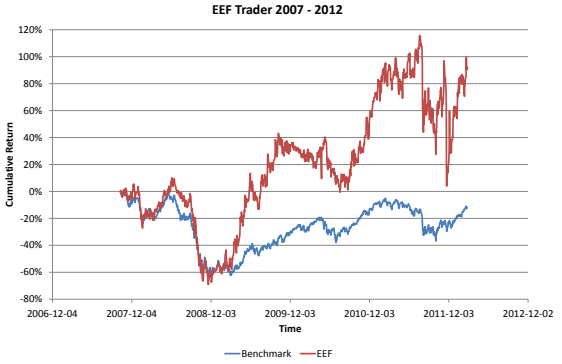
Table 6.16: Scenario results summarized

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	$R_p - R_m$	$max_p - max_m$	$min_p - min_m$
Dow 2003-2012	-1.698E-4	2.7138	-0.8958	0.2297	-0.1942
Dow 2008-2009	1.213E-4	1.7508	-0.0107	0.0331	-0.0467
OBX 2007-2012	1.074E-3	3.0133	1.0423	0.0803	-0.2069
OBX 2008-2009	1.213E-4	1.7508	-0.0107	0.0331	-0.0467

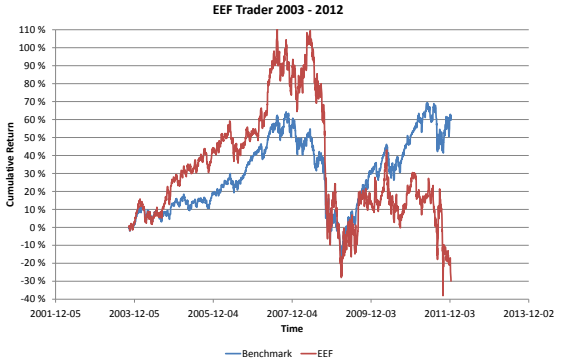
As seen from the variance results in table 6.11 the EEF trader have a substantially higher risk than the benchmark. This is also clear in the test runs seen in figure 6.1a and 6.1b, both these shows a trader with a dangerously high volatility. In both financial crisis scenarios seen in figure 6.1c and 6.1d EFF slightly underperforms compared to the benchmark. In these cases it is less volatile and resembles the index as a whole more closely. The intention with EEF was to combine the defensive approach of a efficient portfolio selector with the more risky stock picking capabilities of the SVM. This approach seems to work on OBX while it fails on Dow. This is again attributed to the level of efficiency on the different markets causing the SVM to little or no predictive ability on Dow, and any it does have drowns in transaction costs there.

6.2.2 Goal Evaluation

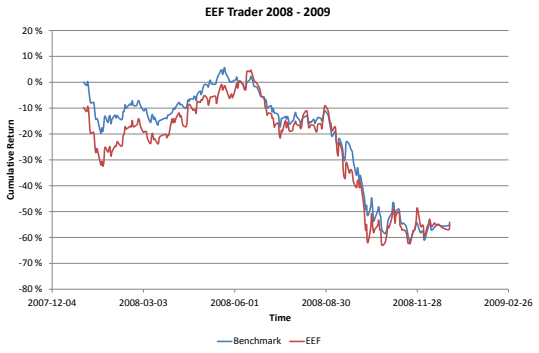
This section revisits the goals set out in section 1.3 and discuss to what degree they are fulfilled.



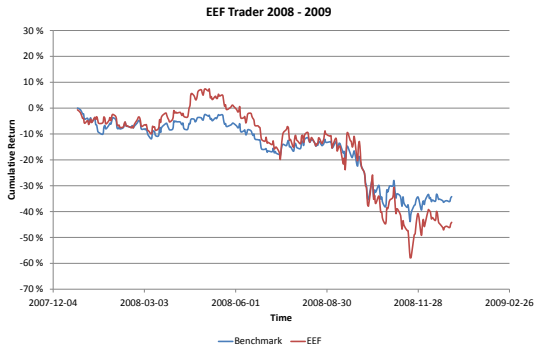
(a) The experimental test run on OBX from January 2007 to February 2012



(b) The experimental test run on Dow from January 2003 to January 2012



(c) The financial crisis on OBX. January 2008 to January 2009



(d) The financial crisis on Dow. January 2008 to January 2009

Figure 6.1: Presentation of EFF runs in different scenarios against the benchmark.

System Architecture

Seamless Module Exchange. The system and its network of nodes ensure seamless exchange of more than just modules, but of any internal part of the trading system. To ease the exchange of large parts the trading module has been introduced. This contains a set of nodes and allows for exchange of larger chunks of functionality with ease. The goal stated that these changes should be available to do post-compilation. This aspect has been disregarded. It was found that there were not worth the effort to make a decent enough interface to support this feature, thus making it easier and more efficient to leave such configuration in the code. The modules and nodes are still easy to put together, but requires compilation afterwards. To retain the ability to test many different approaches at one run the system facilitates for the creation of several trading systems at once, and to run them in parallel from the interface.

Scalability and Parallelization. The nodes operate asynchronously and on separate threads. This allows for parallelization, and the system utilizes all available processor cores it is given. This goes some way towards scalability, but it still limits the system to run on one machine. Through a system of job scheduling and with more general communication interfaces between modules or nodes, the system could be spread on a cluster, but this is far outside of the scope of the four month deadline.

Implemented in Less Than Four Months. The complete system was implemented in a bit more than four months. In man-hours the implementation time did exceed the estimates. An estimated 1500 man hours were used in the coding phase of this project and 25441 KLOC[2] were produced. Tuning of the parameters in the trading systems is not considered as implementation time. This system is scaled a bit down, but in retrospect it should have been scaled much further down to allow for more time to design and test different trading systems.

Technical

Time Series Analysis Methods. No machine learning technique showed any predictive ability on the Dow experiments, though this can be attributed to the efficiency of this market. On OBX however several performed quite well. SVR and SVM performed best as can be seen from the results before transaction costs. The fact that they committed too many trades and incurred heavy transaction costs is irrelevant of their predictive ability. The ANN techniques seem to have some predictive ability though this is much smaller. In all cases it is clear that it requires substantial inefficiencies in the market and as comparison against RSI and MACD shows does it not give better performance than naive technical analysis techniques. On a side note GRNN and PNN was found to be excruciatingly slow, especially on datasets on any size. This makes them harder to test and tune and will limit their use in any live system. Autoregression showed consistently poor performance.

System Training Scheme. Evolution is used as a meta-parameter optimizer, but for many simpler systems (RSI and MACD and FTL) this is the only training they have. In this system the evolution is only run at the start. Many experiments show that the system starts out well, but then degrade in performance the further it gets from the training period. Future plans for the system is to run evolution on a trailing window in the same way as lower level training. This way the systems are continuously tweaked for optimal performance at the cost of some computational performance. There might also be some issues with overfitting and the evolution process must have some mechanic to stop it before this happens in too large a

[2] 1 KLOC = 1000 lines of code

degree. All the classifiers have their own learning mechanisms. Resilient Propagation gives the best general results for MLPNN training and is used to train both the feed-forward and recurrent neural networks. For the GRNN and PNN the conjugate gradient algorithm is used to find the optimal sigma value, while an intelligent modified grid search were used to find the parameters of SVM [41; 61]. Little focus were given to test different SVM and GRNN training methods, rather expert advice were followed.

Robustness and Adaptivity. The training and test datasets are quite different periods. In the Dow case training is bearish, while the test has two bull periods with the financial crisis in the middle. With OBX it trains in a bull period, then runs on the financial crisis and a slight bull period before and after. In the OBX experiments the only systems with decent performance as those that are evolved to buy more or less the whole index and stay in it. This indicates little or no adaptability. In the OBX experiments however some traders like EEF manages to deliver strong results throughout the time period, many of these does however have very high variances. Most are also correlated with the market as the beta values indicate. There are some exceptions Like EEF, RSI and SVM and these can be said to be quite adaptable.

As for robustness any trader that delivers consistent return without too high a variance could be said to be robust as the input data from any financial market can be said to be noisy at best. Some like the ANNs does manage this, and these are the most robust trading systems in this study.

Financial

Input Data. A set of relevant input data had been identified, see section 2.2 for an overview. Many of these were however not tested in the system due to lack of the required data. Tests with fundamental data were rejected from the scope of this project and the focus was set on the mostly technical data that can be created from pure price data. The framework is designed to easily integrate a fundamental data source. One possible conclusion from the tests is that technical data is not enough to identify anomalies that can be capitalized on in a market with transaction costs of the level simulated in these experiments.

Risk and Return. Risk is not a direct part of the fitness function for evolution. There is a possibility of constraining risk, though this was not activated during the experiments. The fitness function uses only average daily returns. The evolutionary training period is 3 years in the Dow Jones case. The results show that such an evolutionary approach is able to incorporate risk. This can be seen most clearly in the example of the "follow-the-leader"(FTL) trader. This chooses how many of the last winners to pick. Even in the case were there are no transaction

costs FTL evolves to pick as many stocks as it is allowed, clearly the most risk averse selection, even though it is only evaluated on return.

One could even argue that a bit too much risk consciousness is present in many trading systems, possibly due to the fact that the training period for Dow is quite bearish. In the experiment with transaction costs on Dow only one trader outperforms the benchmark on returns, while four outperforms it on variance. In the OBX experiments with a more bullish training period traders were less risk averse, though FTL still evolves to pick $n - 1$ of the stocks, which is its upper limit.

It is evident that the system does not need an explicit risk utility function to incorporate risk. Though to appease potential investors the system does have functionality to set a constraint on variance.

Portfolio Evaluation. The framework has several different portfolio evaluation schemes, from the simple approach of picking the n winners in a historical period, to providing a mean-variance optimized portfolio. Early experiments found that a uniform binary vector discrete selection of a set of well performing stocks forming a portfolio is generally better for learning than a more complex non-uniform portfolio scaled on risk or return.

6.2.3 Performance Difference between Dow and OBX

Through the experiments done one the two indices a clear difference in performance is apparent. This section is dedicated to a discussion and analysis of this performance difference and its rationale. To explain the difference four reasons come to mind:

1. The efficiency of the markets
2. Differences in transaction costs schemes
3. The time periods
4. Selection bias

Market Efficiency

Market efficiency might explain the difference in performance. Most traders are intended to find and exploit inefficiencies and naturally this is harder to accomplish in an efficient market. The degree to which the markets are efficient is a result in itself, and to test this is far outside the scope of this project. However, it is natural to assume that one of the worlds most cited and widely recognized indices[51], with stocks from two of the world's largest stock exchanges have a

higher degree of efficiency than a smaller index in Norway. Dow consists of 27 stocks listed at the New York Stock Exchange (NYSE) and 3 from NASDAQ; OBX consists of 25 stocks from Oslo Stock Exchange (OSE). Siow performed a ranking of stock markets according to efficiency. He ranked NYSE 1st, NASDAQ 15th and OSE 20th, there were 25 exchanges in the study[18]. From the experiments done in this project several support the conclusion that OBX is less efficient than Dow. The technical indicators, RSI and MACD, are the most clearly dependent on inefficiencies to perform well. These are the worst performers in the Dow experiments and among the best on OBX.

All in all it seems naturally that some of the rationale for the better performance on OBX stems from a higher degree of inefficiency in that index.

Transaction Cost

The two experiments have different transaction cost schemes. The OBX experiments have fixed transaction costs and use the actual spread data, while the Dow experiments use a relative transaction cost and no spread. To identify the degree to which this affected the difference in outcome from the results without transaction costs, can be compared. An issue here is that all experiments on OBX are done with spread, so there is no way to tell if this has a large impact.

In the Dow experiments only one trader managed a positive cumulative excess return, in the case of OBX 10 of the 14 traders had positive excess returns. Clearly the difference is present without the transaction costs taken into account. The fact that OBX uses spread in this case only further strengthens this conclusion.

While the transaction cost scheme is not the root cause of the difference there are differences in the impact it has. In the OBX case only three traders' performance are degraded by transaction costs, while all traders in the Dow experiments has degraded performance after the application of transaction costs. This might indicate that the scheme for Dow was too harsh or that the one for OBX was too soft. The OBX scheme is a standardized one, previously used for the same dataset several times. The Dow scheme is a conservative cost created by the authors based on the average intraday spread on Dow. As trading at the end of the day usually achieve better spreads through auctions, it can be argued that the Dow transaction cost is set to high.

Time Periods

The two datasets are not of the same time period, though one is a sub-period of the other. The reason for this is that the OBX dataset is to be comparable to other studies by following a standardized setup. The Dow data were taken as far back as was judged to be representative for the modern stock markets. The most

notable difference between the two is that the Dow data has the initial training period in a bearish period where the index as a whole fell 30%. In the training period on OBX from 2006-2007 the index rises by 35%.

Still both datasets have large up and down movements and both start and end at nearly the same value. Both also include the same major event in form of the financial crises in 2008. Further, trading systems are evaluated on their relative performance relative to a benchmark, thus the results are largely independent on the underlying stock market movements. Large volatilities could still have an impact. However, as both datasets have largely the same major periods the difference in time period is not judged to be a source of the difference in results.

Selection Bias

One notable difference between the datasets is the selection of stock data used in the experiments. In the Dow case the companies the index comprised of at the start of the period in question is used, as such no selection bias is introduced. In the OBX case the stocks that were in the index at the end of the period is used. This introduces both the bankruptcy bias as well as a bias in that any stock is guaranteed to have a minimal performance as it is included in the index of the most liquid stocks in the market at the end of the period.

This bias is still unlikely to be the source of the difference due to the way the traders are evaluated. Every evaluation is relative to the benchmark, a benchmark that is subject to the same biases. The bias is thus mitigated by the evaluation scheme.

Conclusion on Performance Difference

To sum up, the difference in performance between the stock markets seems largely to stem from a difference in market efficiency. This explains the performance gap, is supported by the performance of the simple technical traders and backed up by secondary sources. Other factors may have had an impact, but none are found that could have a substantial effect.

6.2.4 Market Anomalies

Contrarian and Momentum Strategy

This project has not discovered any revolutionary technique to predict stock prices. It was however found that many of the more naive technical techniques consistently gave wrong signals. After the discovery all predictors were given a flip option which turned the signal. The option of activating flip was given to evolution. The result of the experiments was that most predictors ended up with this flip signal on. In other words it is preferable to do the opposite of what the predictor predicts.

The rationale behind this can be that when a predictor discovers the stock to be rising, this could actually be a good time to sell. In other words this is the so-called contrarian strategy where one sells stocks that have recently risen and buys those who have fallen. Several studies have found both momentum and contrarian strategies to give significant returns relative to passive trading[44; 103; 127]. Many argue that the both strategies give return though they are opposites. The explanation is that contrarian strategy works on short term trading, while momentum strategies yields returns in the long term [13]. One conclusion to draw from this is the behaviorist notion that the high returns of these simpler strategies are due to an overreaction in the market causing prices to reverse a bit afterwards as a correction. Note that this paper does not have the intention of identifying market efficiency anomalies and thus these evidences were not further investigated. However it can be seen from the results of such simple technical traders, like RSI and MACD, that they have significantly higher returns on OBX, the market though to have the least efficiency.

Other traders' performance can be seen to evaluate the momentum strategy theory. Follow-the-leader (FLT) is essentially only using a momentum strategy as it buys previous winners. While being far from the worst performer this strategy did not give good results, and were far from outperforming the benchmark even without transaction cost. This further strengthens the contrarian argument based on the empirical data from these experiments. The fact that FTL performs better at Dow, while at the same time the technical analysis traders perform much worse further strengthens the evidence that this anomaly is only present in the OBX dataset.

6.2.5 Real World Appliance

Scalability

The results show that efficient frontier traders seems give the best results when risk is considered. In fact, on Dow one can argue that the benchmark seems to be the most attractive investment. This is not a very scalable result as these are mostly buy-and-hold strategies with infrequent rebalancing. Another way to scale trading is to increase the trading resolution; however this is not viable with a buy-and-hold strategy. The only way to scale the returns up is to invest larger initial capital, and though the risk is quite low the returns are not great and on par with most index fund. Alas, there are no great possibilities for quick profit in these results.

Legal Issues

Automated trading is a heavily contended field at the time. Many protest against the amount of automated trading and their impact on the markets. There has been much talk about regulations and rules to limit algorithmic trading. Here is why this should not affect this system:

1. The contention is mostly around HFT, this system trades at a daily basis.
2. Even with a most unlikely full block of algorithmic trading this system would still give value as a portfolio decision support system.
3. Many of the most powerful actors in the field are making money of this trend and are thus unlikely to be willing to abstain from these.

There are no real difference between this system and a stock broker aside from the capability of being automatically connected to the stock market and thus it can be considered to be immune from any potential regulation or rule restricting algorithmic trade. Of course any regulations affecting all trade will still apply.

6.2.6 Technical Discussion

No classifier distinguished themselves in this project, even when given ample opportunity. Most trading systems constructed were based on these techniques. None managed to outperform the efficient portfolio trader without transaction cost; and none did well with transaction costs. The ones that did not have appalling performance with transaction costs were either mimicking the benchmark or sinking heavily into the risk-free-asset. These actions were due to conventional functions like the rebalancer and not from the classifiers themselves. Some pattern recognition and thus price prediction capabilities can be seen in the non-transaction-cost experiments. These were however small and the only success where those that managed excess returns while at the same time reducing the variance. This was in effect only the work of the support vector machine as this is the common denominator of such systems. The SVM is thus considered the best among the classifiers although far from good enough for this domain. The PNN and GRNN were found to be excruciatingly slow when run on the dataset sizes required by evolution to get the half decent results.

Evolution

Evolution was originally planned to be implemented in the same way as other internal learning in this framework. Due to project scope and performance issues this was scaled down to use evolution as a one-shot meta-parameter optimizer. Experiments showed the power of evolution and how well even simple systems like RSI performed for a short time after evolution. In the case of these simpler

technical analysis traders evolution controls every aspect of the trader. This way evolution was able to turn consistently poor performers to those with some of the highest returns.

In the trading systems evolution is used as a meta-parameter optimizer and not as an internal training strategy. These good results from evolution lead to an extension of its capabilities to enabling sliding window evolution. That is evolution that trains regularly in a semi-online manner as were originally planned. This is more time consuming and only tests with the simpler technical traders were done. The results were promising, but further development is required to make it consistent and robust. This is a direction that will be followed further in future work.

As an alternative to evolution, grid search has also been implemented to do more fine grained and complete searches. Evolution is used in most cases as it is faster and as there is seldom need for an optimal solution. The notion of one optimal solution on the training data will likely to be heavily overfitted to that particular data and likely to lack generality to be useful on other data. Evolution allows for a quick and quite good meta-parameter setting for the trading system to run with. As mentioned earlier the evolutionary training also managed to account for risk, even with only return as fitness target.

Learning Supervision

There are several different learning mechanisms in the system. All these are either supervised or reinforced. The only reinforced learning is evolution. To evaluate the fitness of an evolved solution is reinforced learning as it is evaluated only after the simulation is complete. This is easy to evaluate using the average return and the variance over the whole simulation to give a fitness value describing the performance, given the preference for return versus risk. The issue is rather how to evaluate the supervised learning where an agent has to be given continuous feedback on its performance. For the price predictors this was solved by supplying a moving average of returns and possibly the variance of the last n datapoints. There is then a trade-off concerning the length of this time whether one wants to reduce risk or have a reactive trader.

A specific challenge in this system is how to evaluate portfolios with supervised learning. Here the trader has to be presented with an "ideal" portfolio in order to learn how to choose its own portfolios. To construct such an "ideal" portfolio is far from trivial. One option tested in this system is to present the agent with the n best stocks over a period of t and then use this portfolio as ideal in the learning. The exact portfolio presented varies from classifiers to regression agents. With classifiers the n stocks are 1 in the portfolio, while they actually appear as a

normalized portfolio of mean returns when presented to a regression agent. It was found that to present the actual mean did not yield any better results and on the contrary degraded the results. Thus the option with a binary vector was chosen for all agents. By using such a binary vector the agents are predicting whether or not a stock will increase instead of the actual amount it will change.

Portfolio Rebalance

A central component in most trading systems is the rebalancer. This is a node that controls whether or not the trading system should rebalance its portfolio to the new proposed portfolio or stick with the old. The initial rebalancer used the p-norm distance between the portfolios with a threshold to decide whether to rebalance. In this case the expected performance of the new portfolio was not considered, just the distance. This way it was possible to rebalance to a worse portfolio than the current. Initial experiments showed that this often was the case and this approach was discarded before any real experiments were run.

In order to measure the value of the new portfolio in comparable terms to the old a utility function were devised. This were set to $U(r, \sigma) = r/\sigma^2$ and the ratio of U_{new}/U_{old} were measured against a threshold. Many of the traders ended up holding only treasury bonds in turbulent times. The issue is that this give such good utility, as σ were close to 0, that the trader in most case ended up holding the treasury bonds for the rest of the simulation. This resulted in low returns, though also very low variance. Not truly a bad result. Though as the aim of this project were to stochastically outperform the benchmark higher return, and thus less risk aversion were needed. In order to incorporate a measure of risk aversion the utility function were changed. $U(r, \sigma, \gamma) = r - \sigma^2\gamma$ where γ is the risk aversion factor. This is the approach that was used in the real experiments. After the introduction of the latest edition rebalancer there were few issues with traders only holding treasury bonds throughout the simulation.

6.2.7 Potential Issues

Market Simulation

The main weakness of the experiments is the simulated market. Many aspects that incur costs in the real markets are absent in the simulation and not feasible to simulate. The brokerage fee is easy to simulate. The spread between buy and sell is harder, especially without the necessary data as with the Dow dataset in these experiments. This was however present in the OBX dataset. The hardest aspect to simulate is the fact that the trader and other actors affect the market when trading. Especially after the entry of HFTs a price can move substantially from the moment one post an order to when the order gets executed. This effect is

even more present when executing large orders, which this system has no dedicated algorithm to do. This multi-agent aspect is not feasible to model in this system. It is also outside of the scope of this system to handle the game-theoretical aspect of how to post orders. This would require extensive work in itself and is worthy a separate project. In this project it is assumed that the traders have no effect on the market. This assumption is present in other well-known models like the market efficiency theory.

Risk Exposure

Risk exposure was one of the initial worries in this project. It was assumed that machine learning would be able to generate great returns, but at the cost of taking horrendous risk. Still the decision was made to use only return as fitness initially and to adjust this to account directly for variance if needed. To facilitate for risk averse behavior a treasury bond were introduced with daily yields as return (zero in the OBX case). As the results show and as already mentioned in the goal evaluation, there is clear indication that by evolving solution over a longer period of time with only return as fitness, the risk will indirectly be taken into account.

The rationale for this is that any risky trader will in all probability be burned for the risk in the training period given a long enough period. Following from this, the longer the trading period the more risk aversion is inherent in this evolutionary approach. An additional factor is the nature of the training data. The financial market have some long term trends and the training data used in the Dow experiments are from 2000-2003 in which the market as a whole fell somewhat. It is clear from the variance levels that the traders are less risk averse in the OBX experiment where the training period has an upward trend in addition to being three times shorter. Still this should not affect this project as the evaluation always is measured against the benchmark and not on absolute values.

Lack of Fundamental Data

The foundation behind most of the trading systems in this project is to discover and capitalize on anomalies in the stock market. Many of the commonly known anomalies are connected to fundamental data, like the price-earnings ratio. This thesis has built and tested such trading system using only technical data, and it is natural to assume that the systems' ability to identify anomalies is hampered by this. Some anomalies should still be identifiable using the time context and technical indicators, like the weekend effect and over and under reactions. Note that these are just examples and the intention is to discover new and possibly short-lived anomalies and capitalize on them while they still exist. By using data that were able to identify historical anomalies this indicate that it should be sufficient to discover some new ones as well.

6.2.8 Changes From the Fall Project

The Implementation

As with any developmental project one can never be certain how much work is actually required to fulfill the specifications. It was with full knowledge that many parts would never be implemented that the basic design was made in the fall project. During the implementation several redesigns were also done when new and better solutions were discovered. The intention was to implement the system using agile methodologies and as such frequent redesign was required. All in all the resulting system holds high quality and is intended to be developed further and possibly sold in the near future. This additional target of creating a marketable product further increased the implementation time.

Concrete Changes

This is a short list of the most notable concrete changes:

1. Network structure instead of pipeline. Less separated modules, more integrated network
2. More focus on SVM and RBF networks less on MLPNN
3. Less fundamental data, more technical focus due to scope
4. Focus on a smaller set of stocks due to performance
5. Tested against two different data sets
6. Not implemented as a distributed system due to scope

The network architecture was more challenging to implement, but allows for much more freedom in using the framework to create trading systems.

Early experiments confirmed the superior performance of SVM in many aspects and as the project progressed it dawned why most practical systems do not use plain MLPNN, but rather more advanced relatives like the GRNN. As the goal was to create a practical and not purely academic product, the scope was quickly changed to focus more on these techniques.

As mentioned in the goal discussion, retrieving fundamental data is hard and in most cases expensive. A decision was made to not pursue this to a degree that would limit the system implementation and the choice was eventually made to focus on the technical data.

Already in the fall project there were doubts whether the system would be able to handle many stocks when run on the limited hardware available in this project. It

was found that as long as a complete index were used this would supply adequate amounts of data, as well as a good benchmark in the form of the index itself.

To broaden the applicability of the results and to mitigate for the limited amount of stocks tested on it was decided to test it against two different datasets. The indices were chosen for their difference and relevance.

One very ambitious idea was to implement the framework as a distributed system. In the end it was challenging enough to make it threaded and asynchronous. The performance requirements were not as large as initially anticipated and multithreading was more than sufficient. To further make this system distributed could take half a year in itself.

6.3 Conclusion

The results from this thesis conclude that the Dow Jones Industrial Average index is highly efficient. No trading strategy or system able to outperform the uniform buy-and-hold strategy was found in this dataset. The experiments on the OBX index did uncover a degree of market inefficiency. Several trading systems are able to exploit these and generate returns higher than the uniform buy-and-hold strategy. Most trading systems showing excess returns tend to have increased levels of risk. The one that achieves the highest excess return without increasing risk is the efficient frontier trader.

The trading systems are not easily scalable in order to generate more profits. This is because they largely rely on simple buy-and-hold strategies and one cannot simply execute more trades, the only way to scale is to invest more. In non-efficient markets, like OBX, efficient portfolio traders did outperform the benchmark as the benchmark in itself was not efficient. No trading system has however been found that can significantly outperform efficient portfolios.

The results are congruent with modern portfolio theory and support the efficient market hypothesis. The best investment strategy seems to be a mean-variance optimized portfolio. This creates efficient portfolios with low transaction costs. In effect no trader from this system perform better than ordinary index funds.

The machine learning methods used in this thesis show some predictive ability, especially SVM and SVR. Unfortunately, any achieved gains drowned in transaction costs and are not applicable to real markets. Evolution shows promise as a very powerful technique for identifying trading strategies. Evolution with only return as utility seems to be able to incorporate risk if run over a longer time period. This result is most clearly present in the Dow experiments where the evolutionary

training period was three years long.

Future work will apply evolution more directly. The authors will investigate the use of genetic programming to give evolution a more direct and continuous impact. An application interface with a direct connection to markets as well as algorithms for trade execution will also be important in future work as this will greatly reduce the effective transaction costs.

6.4 Contributions

The main goal of this thesis has been to create a working real system, not to pursuit some academic endeavor. Some might say this have led to a project with few or no contributions. On the other hand this project have set to life many theoretical techniques from the fields of artificial intelligence and data mining to solve on one of finance's greatest conundrums, namely how to make money on the stock market. This is a clouded field where there are little incentive to publish what actually works, and where most studies that are published are too academic to be applicable in real life. Though this project also has its issues, by simulating the markets instead of actually trading on them, this is the price for doing academic research. The argument is that this project has given important insight into which techniques and approaches that will work and not work in the real world. Further it is also a contribution to clear a bit up in a field where many of the claims stem from pseudo-science, and even fraud is not uncommon.

6.5 Future Work

This section discusses possible paths to take the system on when this project in done. These are categorized in new inputs, system improvements and new outputs.

6.5.1 Input

More Advanced Input

The input used in this project is assembled from common source that were accessible with limited time and money. In a live implementation it should be strived to identify and acquire unique data that could give the system a competitive edge compared to other systems. Due to the nature of ANNs and its like most any kind of relevant data can easily be utilized and this can be exploited to find patterns other more conventional systems will not be able to contemplate.

Low Resolution Data

In this thesis only daily data have been used. To investigate the possibilities of capitalizing on intraday anomalies more finely granulated data is needed. This data is not likely to be free, which is largely why it is not a part of this project. It could however increase the chances of finding exploitable patterns. With intraday trading possible returns can also be scaled higher than in the current case.

Fundamental Analysis

As the fundamental analysis aspect of the system slipped out of this scope it is one of the most natural steps to improve to the system with. The current system uses only technical data and is as such severely restricted to discover patterns outside those directly connected to price and volume data. A set of useful fundamental indicators are mentioned in this paper as a good starting point, and aside from acquiring a source there is nothing lacking in the framework to include such data. There even where implemented a source that acquired quasi real time fundamental data, though this lacked history and could thus not be used for the experiments done in this project. This source is still not deemed stable enough and for real runs, so there would have to be bought a data source. This cost is the main reason why it is not included in this project.

Sentiment Analysis

A relatively recent trend in investment is to use data mining to find the sentiment of the market by analyzing qualitative data like stock exchange reports or even twitter feeds. This does require some quite advanced AI to do a good analysis. A simple example is to count the number of different words like crisis, recession or bankruptcy. This is exemplified in The Economist's R-word index, where they count the number of newspaper articles with the word "recession" each quarter[5]. Such a sentiment analysis could give the system an edge few others have, especially with a good custom made sentiment algorithm. A sentiment system can also be bought, but is then unlikely to yield much of a competitive advantage as other will use the same.

Other Markets

Due to limited scope this project only tests the trading systems on stock markets. The framework is designed flexible enough that the only thing needed to change markets is to change the output source. Markets could also be combined, though this could complicate the output connection as it might need to interact with several different brokers. The only requirements for these systems to be able to handle a market are access to relevant data and a certain level of liquidity.

6.5.2 System

Online Evolution

In this framework evolution is used as a one-time optimizer and then the system is on its own except for the low level training. Through this work the power of evolution has become clear. There are some issues with overfitting, and some constraints have to be set on the evolution to prevent this. It is still believed that to run evolution semi-online just like the low level training could be advantageous. Initial experiments with RSI and MACD with "sliding evolution" were done and showed promising results. The power of evolution could be further harvested though the use of genetic programming. This entails presenting a tool chest of techniques and operations and let evolution design the actual trading system using these tools. This requires some redesign, but could allow for truly adaptable systems that could change strategy significantly on the run. There could be some issue with robustness though.

High Frequency Trading

There is no doubt HFT is profitable when done right. Though it is way outside both time and monetary budget of the project an idea is to build a HFT system with techniques used in this project. The system would be split in two separate parts. One part of the system would process data to identify patterns using machine learning techniques and genetic algorithms to find strategies to exploit current patterns in the market. Then for each day these strategies would be implemented in a high frequency system with the required data and infrastructure. The strategies would have to be inherently simple in order to run fast enough. The implementation could be done in market downtime as a largely automated process. As with any HFT system this requires considerable capital resources.

Buy/Sell Algorithms

One of the first uses of algorithmic traders was to execute large orders. This requires specific algorithms to be done effectively and to minimize negative effect on the price during the execution. This aspect is not included in this project, but it is an important capability for a real trading system to ensure the cost of transactions keep low and to avoid being exploited by other actors. A significant part of the transaction cost is the inefficiency of the broker, called slippage. By connecting directly to a market without an intermediary this aspect of the transaction cost could largely be eliminated. The algorithms would have to take game theory into account to avoid being exploited by other actors, and could possibly also exploit other traders for profit.

Advanced User Interface

The current system does have a graphic user interface (GUI). This is quite rudimentary and not meant for other than demos and testing by the authors. A though is to extend this to a drag and drop interface where user easily can create and run their own trading systems. Each node with processor will then be represented with a box and one can draw arrows to send data. This could drastically improve the value of the system for less technical potential buyers.

6.5.3 Output

Connect to Real Markets

The current systems are still at the simulation state. By connecting the framework to a market trading framework like TradeStation or NinjaTrader it could actually trade on real markets. A more direct link can be created with more work, but with higher potential margins of return.

When entering a stock market the most natural selection would be Oslo Stock Exchange (OSE). This is due to several factors. It is close both in regards to latency and familiarity as both authors are Norwegian. It has few stocks and reports suggest it might be more inefficient then many larger markets, which makes anomalies more common. For instance Farmen finds that non-trading patterns at OSE gives possibilities for predictability[62]. The results from this paper also indicate inefficiencies and possibilities of excess returns on OSE.

Chapter 7

Appendices

7.1 Mathematical Notation and Concepts

This paper has an extensive use of mathematical notation and concepts that might be useful to review. We assume that all mathematical concepts covered here is known to the reader, and provide this review for reference purposes only.

7.1.1 Vectors and Matrices

Vector and matrix notation is a convenient and succinct way to represent dens data structures and complex operations. They are commonly used in the mathematical branch of linear algebra, though, in thesis they are mainly used for their expressive powers.

Data Structure

A **vector** $\boldsymbol{v} \in \mathbb{R}^n$ represents an ordered list of individual **variables** $v_i \in \mathbb{R}$. The index $i \in \{1, \cdots, n\}$ denotes where a particular variable fits within a vector. By default the variables are considered to be listed vertically as in (7.1). A vector with this configuration is called a column vector. The orientation of the a vector matters when it comes to performing operations on them. Equation (7.2) illustrates the **transpose** operation T, where $\boldsymbol{v}$ is flattened. A neat way to represent a column vector is thus $\boldsymbol{v} = [v_1 \dots v_n]^{\mathrm{T}}$.

$$\boldsymbol{v} = \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} \tag{7.1}$$

$$\boldsymbol{v}^{\mathrm{T}} = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \tag{7.2}$$

An extension of vector notation is the more expressive matrix notation. A matrix $A \in \mathbb{R}^{m \times n}$ is a rectangular array, or table, of single variables $a_{ij} \in \mathbb{R}$. The first index i represents a row in the matrix, whereas the second index j denotes a column. An interpretation is that a matrix is composed of multiple vectors representing either rows (7.3) or columns (7.4). Conversely you could interpret a vector as a matrix with only one column or row, hence the terms column and row vectors.

$$\begin{aligned} A &= \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1^{\mathrm{T}} \\ \vdots \\ \boldsymbol{a}_m^{\mathrm{T}} \end{bmatrix}, \\ \boldsymbol{a}_i^{\mathrm{T}} &= \begin{bmatrix} a_{i1} & \cdots & a_{in} \end{bmatrix}, \quad i \in \{1, \cdots, m\} \end{aligned} \tag{7.3}$$

$$D = \begin{bmatrix} d_{11} & \cdots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{m1} & \cdots & d_{mn} \end{bmatrix} = \begin{bmatrix} \boldsymbol{d}_1 & \cdots & \boldsymbol{d}_n \end{bmatrix}, \quad \boldsymbol{d}_j = \begin{bmatrix} d_{1j} \\ \vdots \\ d_{mj} \end{bmatrix} \tag{7.4}$$

Arithmetic Operations

There are multiple ways to perform multiplication of vectors and matrices. As with simple algebraic notation, two adjacent vectors have an implicit multiplication operation called the **dot product** (7.5). The vectors must be of equal dimensions, where each ordered pair of single variables are multiplied then summed.

$$\boldsymbol{v} \cdot \boldsymbol{u} = \boldsymbol{v}^{\mathrm{T}} \boldsymbol{u} = \begin{bmatrix} v_1 & \cdots & v_n \end{bmatrix} \begin{bmatrix} u_1 \\ \vdots \\ u_n \end{bmatrix} = v_1 u_1 + \cdots + v_n u_n \tag{7.5}$$

General matrix multiplication is an extension, or generalization, of the dot product. Using the dot product, each row vector of the first matrix $A \in \mathbb{R}^{m \times n}$ is multiplied with every column vectors of the second matrix $B \in \mathbb{R}^{n \times k}$. This operation is performed in (7.6). Notice how the inner dimensions n of the matrices must agree. That is, each row vector of matrix A must have the same dimension as the column

vectors of matrix B. The resulting matrix AB is thus of m-by-k dimensions where each entry is defined by (7.7).

$$\begin{aligned} AB &= \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{b}_1 & \cdots & \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{b}_k \\ \vdots & \ddots & \vdots \\ \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{b}_1 & \cdots & \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{b}_k \end{bmatrix} \end{aligned} \tag{7.6}$$

$$\begin{gathered} \boldsymbol{a}_i^{\mathrm{T}}\boldsymbol{b}_j = \begin{bmatrix} a_{i1} & \cdots & a_{in} \end{bmatrix} \begin{bmatrix} b_{1j} \\ \vdots \\ b_{nj} \end{bmatrix} = a_{i1}b_{1j} + \cdots + a_{in}b_{nj}, \\ i \in \{1, \cdots, m\}, \quad j \in \{1, \cdots, k\}, \end{gathered} \tag{7.7}$$

A special case of the matrix multiplication in (7.6) is when $k = 1$, or rather the multiplication of a matrix $A \in \mathbb{R}^{m \times n}$ to a vector $\boldsymbol{b} \in \mathbb{R}^n$. This yields the case (7.8), where the dot product of vector $\boldsymbol{b}$ and each row vector of A produces the vector $A\boldsymbol{b} \in \mathbb{R}^m$.

$$A\boldsymbol{v} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} \boldsymbol{a}_1^{\mathrm{T}}\boldsymbol{v} \\ \vdots \\ \boldsymbol{a}_m^{\mathrm{T}}\boldsymbol{v} \end{bmatrix} = \begin{bmatrix} a_{11}v_1 + \cdots + a_{1n}v_n \\ \vdots \\ a_{m1}v_1 + \cdots + a_{mn}v_n \end{bmatrix} \tag{7.8}$$

In the field of computer science a second more intuitive matrix and vector multiplication is often used. The **entrywise product**, also known as the Hadamard product, applies normal algebraic multiplication of every corresponding entry pairs

of two matrices or vectors of identical dimensions.

$$\begin{aligned} A \circ D &= \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{bmatrix} \circ \begin{bmatrix} d_{11} & \cdots & d_{1n} \\ \vdots & \ddots & \vdots \\ d_{m1} & \cdots & d_{mn} \end{bmatrix} \\ &= \begin{bmatrix} a_{11}d_{11} & \cdots & a_{1n}d_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1}d_{m1} & \cdots & a_{mn}d_{mn} \end{bmatrix} \end{aligned} \tag{7.9}$$

An exception to the dimensionality requirements discussed above is the multiplication of a constant, or weight, to a vector or matrix. In this case the weight is simply multiplied to all entries, as (7.10) illustrates for the vector case. Unsurprisingly, addition and subtraction also behave entrywise as in the vector case in 7.11. Although, for addition and subtraction the dimensions must agree.

$$c\boldsymbol{v} = c \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} = \begin{bmatrix} cv_1 \\ \vdots \\ cv_m \end{bmatrix} \tag{7.10}$$

$$\boldsymbol{x} \pm \boldsymbol{y} = \begin{bmatrix} v_1 \\ \vdots \\ v_m \end{bmatrix} \pm \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} = \begin{bmatrix} v_1 \pm v_1 \\ \vdots \\ v_m \pm v_m \end{bmatrix} \tag{7.11}$$

Vector Norms

An important class of vector and matrix operations is the norm, which is denoted by some variation of $||\cdot||$. Both vectors and matrices have corresponding norm concepts however only norms of vectors are needed. There exists many types of norms, where they all have the common trait of being a measure of length or size of a vector. The most notable norm is the euclidean norm (7.12), often used to calculate the euclidean distance between two points (7.13). A generalization of the euclidean norm is the p-norm defined by (7.14). Besides the euclidean norm, the most important p-norms are the manhattan norm with $p = 1$ (7.15) and the infinity norm with $p = \infty$ (7.16).

$$||\boldsymbol{v}||_2 = \sqrt{\boldsymbol{v}^{\mathrm{T}}\boldsymbol{v}} = \sqrt{v_1^2 + \cdots + v_n^2} \tag{7.12}$$

$$d(\boldsymbol{v}, \boldsymbol{u}) = ||\boldsymbol{v} - \boldsymbol{u}||_2 = \sqrt{(\boldsymbol{v} - \boldsymbol{u})^{\mathrm{T}}(\boldsymbol{v} - \boldsymbol{u})} = \sqrt{\sum_{i=1}^{n} (v_i - u_i)^2} \tag{7.13}$$

$$||\boldsymbol{v}||_1 = \left(\sum_{i=1}^{n} v_i^p \right)^{\frac{1}{p}} \tag{7.14}$$

$$||\boldsymbol{v}||_1 = \left(\sum_{i=1}^{n} v_i^1 \right)^{\frac{1}{1}} = \sum_{i=1}^{n} |v_i| \tag{7.15}$$

$$||\boldsymbol{v}||_\infty = \max\{v_1, \cdots, v_n\}; \tag{7.16}$$

Linear and Quadratic Equations

Matrix notation allows for succinct expression of linear and quadratic systems of equations. The set of linear equations $y_i = a_{i1}x_{i1} + \cdots + a_{in}x_{in} + b_i = 0,\ i \in \{1, \cdots, m\}$ can be written as (7.17). The quadratic equation $y = \sum_{i,j=1}^{n} a_{ij}x_i x_j + b = 0$ can be written as (7.18).

$$\boldsymbol{y} = A\boldsymbol{x} + \boldsymbol{b} = \boldsymbol{0}; \tag{7.17}$$

$$y = \boldsymbol{x}^{\mathrm{T}} A \boldsymbol{x} + b = 0; \tag{7.18}$$

Miscellaneous Structures and Operations

Sometimes it is useful to have a way to represent a vector of identical entries. The null vectors $\boldsymbol{0}$ and one vectors $\boldsymbol{1}$ are vectors with entries of only 0 and 1 respectively. Usually the dimension of such a vector is implied by the context. An application of the one vector is that it can be used to sum the entries of a vector by $\boldsymbol{1}^{\mathrm{T}}\boldsymbol{x} = x_1 + \cdots + x_m$.

$$\boldsymbol{0} = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}, \quad \boldsymbol{1} = \begin{bmatrix} 1 \\ \vdots \\ 1 \end{bmatrix} \tag{7.19}$$

The diag(·) operator transforms a vector $\boldsymbol{v} \in \mathbb{R}^n$ into a diagonal matrix $V \in \mathbb{R}^{n \times n}$ by putting each entry of the vector into the diagonal of the matrix while keeping other entries zero.

$$\text{diag}(\boldsymbol{v}) = \text{diag}\begin{bmatrix} v_1 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} v_1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & v_n \end{bmatrix} = V \tag{7.20}$$

An often used matrix concept is the identity matrix defined as $I_n = diag(\mathbf{1})$. It has the property that multiplication with it does not have any effect, e.g., $AI_n = A$, and can be viewed as a cousin of 1 in normal arithmetic.

Finally augmentation of vectors and matrices can be needed for application of machine learning and optimization software. Augmentation simply means that we concatenate two object of equal dimensions. For instance, a vector $\boldsymbol{v} = [v_1, \cdots, v_n]^{\mathrm{T}}$ may be augmented by a new variable v_{n+1} becoming $\boldsymbol{v}' = [v_1, \cdots, v_{n+1}]^{\mathrm{T}}$. Similarly a matrix $A \in \mathbb{R}^{m \times n}$ may be augmented another with, say, matrix $E \in \mathbb{R}^{m \times k}$ as in (7.21) or vector as in (7.22).

$$\begin{bmatrix} A & E \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} & e_{11} & \cdots & e_{1k} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} & e_{m1} & \cdots & e_{mk} \end{bmatrix} \tag{7.21}$$

$$\begin{bmatrix} A \\ \boldsymbol{v}^{\mathrm{T}} \end{bmatrix} = \begin{bmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \\ v_1 & \cdots & v_n \end{bmatrix} \tag{7.22}$$

7.2 Technical SVM Details

The basic SVM classifier assumes datapoint separability. It has a one-augmented dataset $X \in \mathbb{R}^{m \times n}$ of n datapoints $\boldsymbol{x}_i$ and m features (including the augmented feature). This follows the dataset definition of (2.35), where X is a matrix with rows of datapoints $\boldsymbol{x}_i$. Since this is not a multi-label problem there is only one label per datapoint. One can therefore compound all the labels related to dataset X into one single binary vector $[y_1 \ \cdots \ y_m]^{\mathrm{T}} = \boldsymbol{y} \in \{-1, 1\}^m$ which we want to use

to find the model parameters $\boldsymbol{\theta}$. The separating hyperplane is defined by $\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{z} = 0$ for any $\boldsymbol{z} = [1 \quad z_2 \ \cdots \ z_n]^{\mathrm{T}} \in \mathbb{R}^n$. Surrounding this we want to make two parallel separating hyperplanes, or margins, which we maximize the distance between. For the two different classification we can defined the margins by (7.23), which can be combined to that of (7.24).

$$\begin{aligned} \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_i &\leq +1 \quad \text{for} \quad y_i = +1 \\ \boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_i &\geq -1 \quad \text{for} \quad y_i = -1 \end{aligned} \tag{7.23}$$

$$y_i\boldsymbol{\theta}^{\mathrm{T}}\boldsymbol{x}_i \geq 1 \ \forall i \tag{7.24}$$

Because the parameter vector $\boldsymbol{\theta}$ is the normal to the separating hyperplane we have that $\frac{1}{||\boldsymbol{\theta}||_2}$ is the "width" of the margin. This means we can maximize this margin by minimizing the denominator, i.e., Euclidean norm $||\boldsymbol{\theta}||_2$, subject to the separation of all the datapoints. This leads to the raw minimization problem of (7.25). Here, the function $\mathrm{diag}(\cdot)$ converts a n dimensional vector into a n-by-n matrix with the vector entries in the diagonal, otherwise zero. To heuristically solve this we can cast it into a Lagrangian relaxed quadratic optimization problem[38]. It can be shown that we are trying to find the saddle point of (7.26). The reason for finding the saddle point rather than the minimum is that the Lagrangian multipliers will otherwise get the trivial solution of $\lambda_i = \inf$. Datapoints $\boldsymbol{x}_i$ that have Lagrangian multipliers counterparts which are $\lambda_i > 0$ are said to be the support vectors. They are the once that define the separating hyperplane.

$$\begin{aligned} \min_{\boldsymbol{\theta}} \quad & ||\boldsymbol{\theta}||_2 \\ \text{s.t.} \quad & \mathrm{diag}(\boldsymbol{y})(X\boldsymbol{\theta} - \mathbf{1}) \geq \mathbf{0} \end{aligned} \tag{7.25}$$

$$\begin{aligned} \min_{\boldsymbol{\theta}} \max_{\boldsymbol{\lambda}} \quad & \tfrac{1}{2}\boldsymbol{\theta}^2 - \boldsymbol{\lambda}^{\mathrm{T}} \left[\mathrm{diag}(\boldsymbol{y})(X\boldsymbol{\theta} - \mathbf{1})\right] \\ \text{s.t.} \quad & \boldsymbol{\lambda} \geq 0 \end{aligned} \tag{7.26}$$

An expansion of the formulation of (7.25) can be made where a soft margins is included. It introduces a variable $\boldsymbol{\xi} \in \mathbb{R}^n$ yielding (7.27). As done with the basic version this can be Lagrange relaxed and made into a quadratic optimization problem. Constant C is a regularization meta-parameter that must be tuned. It reflects the trade-off between maximizing the margin and minimizing the training

error.

$$\begin{aligned} \min_{\boldsymbol{\theta},\boldsymbol{\xi}} \quad & ||\boldsymbol{\theta}||_2 + C\mathbf{1}^{\mathrm{T}}\boldsymbol{\xi} \\ \text{s.t.} \quad & \boldsymbol{\xi} \geq \mathrm{diag}(\boldsymbol{y})(X\boldsymbol{\theta} - \mathbf{1}),\ \boldsymbol{\xi} \geq \mathbf{0} \end{aligned} \tag{7.27}$$

7.3 Additional Results

Here is the results from the training part of the experiments presented.

Table 7.1: Results from training runs without transaction cost on DOW from 2000-2003

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	R_p/R_m	$max_p - max_m$	$min_p - min_m$
AR	1.165E-4	1.3825	0.0883	0.0398	0.02
MLPNN	9.201E-5	0.5939	0.1535	-8.188E-3	0.0252
GRNN	-8.231E-6	1.0702	0.0439	2.183E-3	-2.591E-3
PNN	7.314E-5	0.7002	0.1028	-1.335E-4	0.0177
SVM	5.04E-4	1.3056	0.3821	0.0158	-2.615E-3
SVR	4.441E-4	0.2806	0.4418	-0.0213	0.0352
MHS	1.505E-4	0.9178	0.1097	-1.17E-3	-0.0113
SM	2.121E-4	1.0268	0.1497	-1.041E-3	0.0211
EF	3.826E-4	0.8709	0.2658	-0.0135	-0.0155
REF	4.815E-4	1.5082	0.3613	0.0315	0.0217
EEF	3.693E-4	1.0524	0.3594	0.0141	0.0161
RSI	-1.026E-4	4.7857	-1.3264	0.1902	-0.1654
MACD	2.225E-4	2.7707	-0.0352	0.1322	-0.1229
FTL	7.433E-7	0.9915	0.0925	-3.36E-3	1.215E-3

In tables 7.2 and 7.1 the results from the Dow experiments' training runs is presented. This is presented to illustrate the degree to which the trading system manages to fit data. Compared with the test results this gives a good indication of the generalizability of the solution and to what degree it is overfitted.

Tables 7.3 and 7.4 shows the training results from the OBX experiments.

Table 7.2: Results from training runs with transaction cost on DOW from 2000-2003

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	R_p/R_m	$max_p - max_m$	$min_p - min_m$
AR	-2.339E-4	1.6242	-0.1919	0.0252	-0.0134
MLPNN	6.849E-5	0.7892	0.1088	-1.245E-4	0.0139
GRNN	2.615E-4	5.504E-6	0.2683	-0.0632	0.0751
PNN	-5.753E-4	0.8313	-0.2609	-0.0118	0.0212
SVM	1.64E-4	1.7527	0.1415	0.0234	-0.0301
SVR	9.39E-4	0.5679	0.8129	-0.0178	7.396E-3
MHS	2.589E-4	0.8438	0.2064	-4.638E-3	0.0108
SM	1.88E-4	2.8641	0.0898	0.0488	-0.0259
EF	8.129E-4	2.4681	0.4997	0.0935	-0.0219
REF	2.116E-4	1.6606	0.1272	0.0161	-0.0202
EEF	1.113E-4	1.7833	0.0506	0.0318	-0.0186
RSI	-6.059E-4	12.8522	-1.8832	0.4813	-0.3054
MACD	-2.218E-4	7.3929	-1.6602	0.2772	-0.3054
FTL	-1.268E-4	0.9915	0.0137	-3.498E-3	1.077E-3

Table 7.3: Results from training runs without transaction cost on OBX from 2006-2007

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	R_p/R_m	$max_p - max_m$	$min_p - min_m$
AR	1.261E-3	4.7206	0.1365	0.0596	-0.0492
MLPNN	-2.444E-4	1.0626	-8.091E-3	-3.108E-4	7.618E-4
GRNN	9.701E-5	1.1025	0.0116	7.969E-4	-1.319E-3
PNN	1.638E-4	1.3491	0.0169	8.189E-3	-4.273E-3
SVM	1.679E-3	4.8624	0.2496	0.0596	-0.0492
SVR	2.158E-3	5.1972	0.0799	0.0284	-0.0184
MHS	-9.53E-4	1.2378	-0.074	2.372E-3	-7.304E-3
SM	1.197E-3	3.6145	0.1785	0.0275	-0.0399
EF	6.701E-4	1.8568	0.0776	0.0139	4.084E-3
REF	1.226E-3	1.5427	0.0945	9.379E-3	-4.272E-3
EEF	2.611E-3	2.1945	0.1711	0.0204	1.325E-3
RSI	2.068E-3	5.7498	0.6661	0.2384	-0.0529
MACD	1.407E-3	4.3496	0.4038	0.0728	-0.0567
FTL	-3.253E-4	0.9836	-0.1146	-1.653E-3	-1.677E-3

Table 7.4: Results from training runs with transaction cost on OBX from 2006-2007

Trader	$\bar{r}_p - \bar{r}_m$	σ_p^2/σ_m^2	R_p/R_m	$max_p - max_m$	$min_p - min_m$
AR	1.261E-3	4.7206	0.1365	0.0596	-0.0492
MLPNN	-2.652E-4	1.0623	-9.486E-3	-3.108E-4	7.618E-4
GRNN	5.115E-4	1.4572	0.0529	5.482E-3	-6.961E-3
PNN	1.638E-4	1.3491	0.0169	8.189E-3	-4.273E-3
SVM	1.679E-3	4.8624	0.2496	0.0596	-0.0492
SVR	2.142E-3	5.1957	0.0792	0.0284	-0.0184
MHS	-4.076E-3	1.47	-0.241	5.1E-3	-0.0138
SM	-2.921E-4	1.2686	-0.0213	4.179E-3	-2.371E-3
EF	3.377E-3	5.0385	1.1793	0.2384	-0.0396
REF	6.706E-4	1.4999	0.0589	8.971E-3	-4.272E-3
EEF	2.611E-3	2.1945	0.1711	0.0204	1.325E-3
RSI	2.044E-3	5.7468	0.6542	0.2384	-0.0529
MACD	1.389E-3	4.3495	0.3959	0.0728	-0.0567
FTL	-8.84E-4	0.9821	-0.2818	-2.278E-3	-2.236E-3

Bibliography

[1] Oxford English Dictionary. URL http://oxforddictionaries.com/.

[2] The sveriges riksbank prize in economic sciences in memory of alfred nobel 1990, 1990. URL http://www.nobelprize.org/nobel_prizes/economics/laureates/1990/.

[3] ojalgo, 2007. URL http://ojalgo.org.

[4] Copper, red bull, the world's most informative metal. *The Economist*, 2011. URL http://www.economist.com/node/21530107.

[5] The r-word index, up means down. *The Economist*, 2011. URL http://www.economist.com/node/21529079.

[6] High-frequency trading, the fast and the furious. *The Economist*, 2012. URL http://www.economist.com/node/21547988.

[7] Børsen angriper roboter med avgift, 2012. URL http://stocklink.no/Article.aspx?id=93016.

[8] J S Abarbanell and B J Bushee. Fundamental analysis, future earnings, and stock prices. *Journal of Accounting Research*, 35(1):1–24, 1997. URL http://www.jstor.org/stable/2491464.

[9] J S Abarbanell and B J Bushee. Abnormal returns to a fundamental analysis strategy. *Accounting Review*, 73(1):19–45, 1998. URL http://www.jstor.org/stable/248340.

[10] Steven B. Achelis. *Technical Analysis from A to Z*. Probus Publishing, Chicago, 1995.

[11] Monica Adya and Fred Collopy. How effective are neural networks at forecasting and prediction? a review and evaluation. *Journal of Forecasting*, 17(5-6):481–495, 1998. ISSN 1099-131X.

doi: 10.1002/(SICI)1099-131X(1998090)17:5/6⟨481::AID-FOR709⟩3.0.CO;2-Q. URL http://dx.doi.org/10.1002/(SICI)1099-131X(1998090)17:5/6<481::AID-FOR709>3.0.CO;2-Q.

[12] John Affleck-Graves, Larry R Davis, and Richard R Mendenhall. Forecasts of earnings per share: Possible sources of analyst superiority and bias. *Contemporary Accounting Research*, 6(2):501–517, 1990. URL http://doi.wiley.com/10.1111/j.1911-3846.1990.tb00771.x.

[13] Manuel Ammann, Marcel Moellenbeck, and Markus M Schmid. Contrarian and momentum strategies in the spanish stock market. *Journal of Asset Management*, 9(1):362–374, 2003. URL http://www.palgrave-journals.com/doifinder/10.1057/jam.2010.22.

[14] Andersen and Mikelsen. Algorithmic portfolio optimisation using evolutionary algorithms on neural networks and supporting classifiers. 2011.

[15] P.J. Angeline, G.M. Saunders, and J.B. Pollack. An evolutionary algorithm that constructs recurrent neural networks. *Neural Networks, IEEE Transactions on*, 5(1):54 –65, jan 1994. ISSN 1045-9227. doi: 10.1109/72.265960.

[16] G. Appel. *Technical Analysis: Power Tools for Active Investors*. Financial Times Prentice Hall books. Financial Times/Prentice Hall, 2005. ISBN 9780131479029. URL http://books.google.co.in/books?id=RFYIAAAACAAJ.

[17] G. Armano, M. Marchesi, and a. Murru. A hybrid genetic-neural architecture for stock indexes forecasting. *Information Sciences*, 170(1):3–33, Feb 2005. doi: 10.1016/j.ins.2003.03.023. URL http://linkinghub.elsevier.com/retrieve/pii/S002002550300433X.

[18] Michael J. Aitken Audris S. Siow. Ranking world equity markets on the basis of market efficiency and integrity. (5), 2003.

[19] Donald B. and Keim. Size-related anomalies and stock return seasonality: Further empirical evidence. *Journal of Financial Economics*, 12(1):13 – 32, 1983. ISSN 0304-405X. doi: 10.1016/0304-405X(83)90025-9. URL http://www.sciencedirect.com/science/article/pii/0304405X83900259.

[20] K Reid B Rosenberg. Persuasive evidence of market inefficiency. 1998.

[21] Bahman and Kermanshahi. Recurrent neural network for forecasting next 10 years loads of nine japanese utilities. *Neurocomputing*, 23(1-3):125 – 133, 1998. ISSN 0925-2312. doi: 10.1016/S0925-2312(98)00073-3. URL http://www.sciencedirect.com/science/article/pii/S0925231298000733.

[22] Jonas Blich Bakken. Femøring skal temme aksje-roboter, 2012. URL http://www.dagensit.no/article2402979.ece.

[23] Arun Bansal, Robert J. Kauffman, and Rob R. Weitz. Comparing the modeling performance of regression and neural networks as data quality varies: a business value approach. *J. Manage. Inf. Syst.*, 10:11–32, June 1993. ISSN 0742-1222. URL http://dl.acm.org/citation.cfm?id=1189680.1189683.

[24] Zhejing Bao, Daoying Pi, and Youxian Sun. Short-term load forecasting based on self-organizing map and support vector machine. In Lipo Wang, Ke Chen, and Yew Ong, editors, *Advances in Natural Computation*, volume 3610 of *Lecture Notes in Computer Science*, pages 419–419. Springer Berlin / Heidelberg, 2005. ISBN 978-3-540-28323-2. URL http://dx.doi.org/10.1007/11539087_89.

[25] Nicholas Barberis and Richard Thaler. Chapter 18 a survey of behavioral finance. In M. Harris G.M. Constantinides and R.M. Stulz, editors, *Financial Markets and Asset Pricing*, volume 1, Part B of *Handbook of the Economics of Finance*, pages 1053 – 1128. Elsevier, 2003. doi: 10.1016/S1574-0102(03)01027-6. URL http://www.sciencedirect.com/science/article/pii/S1574010203010276.

[26] Dean Sherman Barr. Predictive neural network means and method for selecting a portfolio of securities wherein each network has been trained using data relating to a corresponding security, 1998. URL http://www.google.no/patents/US5761442.pdf?source=gbs_overview_r&cad=0.

[27] S. Basu. Investment performance of common stocks in relation to their price-earnings ratios: A test of the efficient market hypothesis. *The Journal of Finance*, 32(3):pp. 663–682, 1977. ISSN 00221082. URL http://www.jstor.org/stable/2326304.

[28] Meredith Beechey, David Gruen, and James Vickery. The efficient market hypothesis: A survey. RBA Research Discussion Papers rdp2000-01, Reserve Bank of Australia, January 2000. URL http://ideas.repec.org/p/rba/rbardp/rdp2000-01.html.

[29] Richard K. Belew, John McInerney, and Nicol N. Schraudolph. Evolving Networks: Using the Genetic Algorithm with Connectionist Learning. In Christopher G. Langton, Charles Taylor, J. Doyne Farmer, and Steen Rasmussen, editors, *Artificial Life II*, volume 10 of *SFI Studies in the Sciences*

of Complexity: Proceedings, pages 511–547. Addison-Wesley, Redwood City, CA, 1992.

[30] Sidney Cottle Roger F. Murray Frank E. Block Benjamin Graham, David Le Fevre Dodd. *Graham and Dodd's security analysis*. McGraw-Hill Professional, 1951.

[31] Hendrik Bessembinder. Bid-ask spreads : Measuring trade execution costs in financial markets. 2009.

[32] Bruno Biais. High frequency trading, 2011. URL `http://www.eifr.eu/files/file8379010.pdf`.

[33] Werner F. M. De Bondt and Richard Thaler. Does the stock market overreact? *The Journal of Finance*, 40(3):pp. 793–805, 1985. ISSN 00221082. URL `http://www.jstor.org/stable/2327804`.

[34] R B Boozarjomehry and W Y Svrcek. Automatic design of neural network structures. *Computers & Chemical Engineering*, 25(7-8): 1075–1088, 2001. URL `http://linkinghub.elsevier.com/retrieve/pii/S0098135401006809`.

[35] Jonathan A Brogaard. High frequency trading and its impact on market quality. *Management*, page 66, 2010. URL `http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1641387`.

[36] James H. Brookmire. Methods of business forecasting based on fundamental statistics. *The American Economic Review*, 3(1):pp. 43–58, 1913. ISSN 00028282. URL `http://www.jstor.org/stable/1828258`.

[37] Philip Brown, Allan W. Kleidon, and Terry A. Marsh. New evidence on the nature of size-related anomalies in stock prices. *Journal of Financial Economics*, 12(1):33 – 56, 1983. ISSN 0304-405X. doi: 10.1016/0304-405X(83)90026-0. URL `http://www.sciencedirect.com/science/article/pii/0304405X83900260`.

[38] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998. ISSN 1384-5810. URL `http://dx.doi.org/10.1023/A:1009715923555`. 10.1023/A:1009715923555.

[39] Jeff Castura, Robert Litzenberger, Richard Gorelick, and Yogesh Dwivedi. Market efficiency and microstructure evolution in u . s . equity markets : A high-frequency perspective. *Russell The Journal Of The Bertrand Russell Archives*, page 24, 2010.

[40] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning, And Games*. Cambridge University Press, 2006. ISBN 9780521841085. URL http://books.google.no/books?id=zDnRBlazhfYC.

[41] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[42] Peter D Chant. On the predictability of corporate earnings per share behavior. *Journal of Finance*, 35(1):13–21, 1980. URL http://www.jstor.org/stable/2327177?origin=crossref.

[43] C S Agnes Cheng and Ray McNamara. The valuation accuracy of the price-earnings and price-book benchmark valuation methods. *Review of Quantitative Finance and Accounting*, 15(4):349–370, 2000. URL http://www.springerlink.com/index/G824008911251411.pdf.

[44] Pin-Huang Chou, K C John Wei, and Huimin Chung. Sources of contrarian profits in the japanese stock market. *Journal of Empirical Finance*, 14(3):261–286, 2007. URL http://linkinghub.elsevier.com/retrieve/pii/S0927539806000697.

[45] Y. Chou. *Statistical Analysis: With Business and Economic Applications*. Holt, Rinehart & Winston, 1975. URL http://books.google.no/books?id=zGavugAACAAJ.

[46] P R Cohen and A E Howe. How evaluation guides ai research: The message still counts more than the medium. *AI Magazine*, 9(4):35, 1988. URL http://www.aaai.org/ojs/index.php/aimagazine/article/viewArticle/952.

[47] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995. ISSN 0885-6125. URL http://dx.doi.org/10.1007/BF00994018. 10.1007/BF00994018.

[48] Gerben Driesprong, Ben Jacobsen, and Benjamin Maat. Striking oil: Another puzzle? *Journal of Financial Economics*, 89(2):307–327, 2008. URL http://linkinghub.elsevier.com/retrieve/pii/S0304405X08000901.

[49] Daniel W. Dyer. Watchmaker framework for evolutionary computation, 2006. URL http://watchmaker.uncommons.org/.

[50] Peter D. Easton. PE Ratios, PEG Ratios, and Estimating the Implied Expected Rate of Return on Equity Capital. *SSRN eLibrary*, 2003. doi: 10.2139/ssrn.423601.

[51] R. Edelman. *The Truth about Money*. Rodale, 2005. ISBN 9781594861642. URL http://books.google.no/books?id=_XNN7-MyOp0C.

[52] Charles Elkan. Predictive analytics and data mining, 2011. URL http://cseweb.ucsd.edu/users/elkan/291/dm.pdf.

[53] Charles Elkan. Maximum likelihood, logistic regression,and stochastic gradient training, 2011. URL http://cseweb.ucsd.edu/~elkan/250B/logreg.pdf.

[54] Jeffrey L. Elman. Finding structure in time. *Cognitive Science*, 14(2):179 – 211, 1990. ISSN 0364-0213. doi: 10.1016/0364-0213(90)90002-E. URL http://www.sciencedirect.com/science/article/pii/036402139090002E.

[55] Frank J Fabozzi, Sergio M Focardi, and Caroline Jonas. High-frequency trading: methodologies and market impact. *Exchange Organizational Behavior Teaching Journal*, 19(1):7–37, 2011. URL http://www.conatum.com/presscites/HFTMMI.pdf.

[56] Eugene Fama. Fama on Market Efficiency in a Volatile Market. URL http://www.dimensional.com/famafrench/2009/08/fama-on-market-efficiency-in-a-volatile-market.html.

[57] Eugene F. Fama. The behavior of stock-market prices. *The Journal of Business*, 38(1):pp. 34–105, 1965. ISSN 00219398. URL http://www.jstor.org/stable/2350752.

[58] Eugene F. Fama. Random walks in stock market prices. *Financial Analysts Journal*, 21(5):pp. 55–59, 1965. ISSN 0015198X. URL http://www.jstor.org/stable/4469865.

[59] Eugene F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):pp. 383–417, 1970. ISSN 00221082. URL http://www.jstor.org/stable/2325486.

[60] Eugene F Fama and Kenneth R French. Dividend yields and expected stock returns. *Journal of Financial Economics*, 22(1):3–25, 1988. URL http://www.sciencedirect.com/science/article/pii/0304405X88900207.

[61] Rong-en Fan, Pai-hsuen Chen, and Chih-jen Lin. Working set selection using second order information for training support vector machines. *Journal of Machine Learning Research*, 6(6):1889–1918, 2005. URL http://portal.acm.org/citation.cfm?id=1194907.

[62] Tom E. S. Farmen. Return predictability from thin trading? evidence from oslo stock exchange.

[63] D.B. Fogel. An introduction to simulated evolutionary optimization. *Neural Networks, IEEE Transactions on*, 5(1):3 –14, jan 1994. ISSN 1045-9227. doi: 10.1109/72.265956.

[64] R J Frank, N Davey, and S P Hunt. Time series prediction and neural networks. 2001. URL http://www.google.no/url?sa=t&rct=j&q=timeseriespredictionandneuralnetworksr.j.frank%2Cn.davey%2Cs.p.hunt&source=web&cd=1&ved=0CCIQFjAA&url=http%3A%2F%2Fwww.smartquant.com%2Freferences%2FNeuralNetworks%2Fneural30.pdf&ei=sMvgTuHnEM744QS_n5yIDQ&usg=AFQjCNHhYX3R77UXhdhu2NjVoJ_hWw6DDQ&sig2=MTRz2nmJXPVJsonrd5iljQ.

[65] Lee N . Price Frank A . Sortino. Performance measurement in a downside risk framework. *The Journal of Investing*, 3(3):pp. 59–64, 1994.

[66] George M Frankfurter and Elton G McGoun. Resistance is futile: the assimilation of behavioral finance. *Journal of Economic Behavior and Organization*, 48(4):375 – 389, 2002. ISSN 0167-2681. doi: 10.1016/S0167-2681(01)00241-4. URL http://www.sciencedirect.com/science/article/pii/S0167268101002414.

[67] Craig W. French. Jack Treynor's 'Toward a Theory of Market Value of Risky Assets'. *SSRN eLibrary*, 2002.

[68] Kenneth R French. Stock returns and the weekend effect. *Journal of Financial Economics*, 8(1):55–69, 1980. URL http://www.sciencedirect.com/science/article/pii/0304405X80900215.

[69] Alexei A. Gaivoronski and Georg C. Pflug. Value-at-risk in Portfolio Optimization: Properties and Computational Approach. *SSRN eLibrary*, 2004.

[70] Alexei A Gaivoronski, Nico Van Der Wijst, and Sergiy Krylov. Optimal portfolio selection and dynamic benchmark tracking. *European Journal of Operational Research*, 163(1):115–131, May 2005. doi: 10.1016/j.ejor.2003.12.001. URL http://linkinghub.elsevier.com/retrieve/pii/S0377221703009123.

[71] C.L. Giles and Steve Lawrence. Noisy time series prediction using recurrent neural networks and grammatical inference. pages 161–183, 2001.

[72] C.L. Giles, S. Lawrence, and Ah Chung Tsoi. Rule inference for financial prediction using recurrent neural networks. In *Computational Intelligence for Financial Engineering (CIFEr), 1997., Proceedings of the IEEE/IAFE 1997*, pages 253 –259, mar 1997. doi: 10.1109/CIFER.1997.618945.

[73] B. Graham, D. Dodd, and D.L. Dodd. *Security Analysis: The Classic 1940 Edition*. McGraw-Hill book company. McGraw-Hill, 2002. ISBN 9780071412285. URL http://books.google.no/books?id=pDBZ0j4Vc-4C.

[74] Massimo Guidolin, Stuart Hyde, David McMillan, and Sadayuki Ono. Non-linear predictability in stock and bond returns: When and where is it exploitable? *International Journal of Forecasting*, 25(2):373–399, 2009. URL http://www.sciencedirect.com/science/article/B6V92-4VR6K5X-1/2/f7aaf36ea4d526a70e6ab05583c97911.

[75] Andrew G Haldane. Patience and finance, 2010. URL http://www.bankofengland.co.uk/publications/speeches/2010/speech445.pdf.

[76] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software. *ACM SIGKDD Explorations Newsletter*, 11(1):10, 2009. URL http://doi.acm.org/10.1145/1656274.1656278.

[77] Joel Hasbrouck and Gideon Saar. Low-latency trading. *New York*, 10012: 57, 2010. URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1695460.

[78] M Hassan, B Nath, and M Kirley. A fusion model of hmm, ann and ga for stock market forecasting. *Expert Systems with Applications*, 33(1):171–180, Jul 2007. doi: 10.1016/j.eswa.2006.04.007. URL http://linkinghub.elsevier.com/retrieve/pii/S0957417406001291.

[79] Mark Haug and Mark Hirschey. The January Effect. *SSRN eLibrary*, 2005.

[80] Robert A. Haugen and Philippe Jorion. The january effect: Still there after all these years. *Financial Analysts Journal*, 52(1):pp. 27–31, 1996. ISSN 0015198X. URL http://www.jstor.org/stable/4479893.

[81] Jeff Heaton. Encog artificial intelligence framework for java and dotnet, 2008. URL http://www.heatonresearch.com/encog.

[82] Terrence Hendershott and Pamela C Moulton. Automation, speed, and stock market quality: The nyses hybrid. *Journal of Financial Markets*, 14(4): 568–604, 2011. URL http://linkinghub.elsevier.com/retrieve/pii/S138641811100005X.

[83] Terrence Hendershott, Charles M Jones, and Albert J Menkveld. Does algorithmic trading improve liquidity ? *Journal of Finance*, 66(1): 1–33, 2011. URL http://onlinelibrary.wiley.com/doi/10.1111/j.1540-6261.2010.01624.x/full.

[84] C Igel and M Hüsken. Improving the rprop learning algorithm. *Proceedings of the second international ICSC*, pages 115–121, 2000. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.17.3899&rep=rep1&type=pdf.

[85] Robert Jarrow and Philip Protter. A dysfunctional role of high frequency trading in electronic markets. *New York*, (March):1–13, 2011. URL http://papers.ssrn.com/sol3/papers.cfm?abstract_id=1781124.

[86] Won Chul Jhee and Jae Kyu Lee. Performance of neural networks in managerial forecasting. *Int. J. Intell. Syst. Account. Financ. Manage.*, 2:55–71, January 1993. ISSN 1055-615X. URL http://dl.acm.org/citation.cfm?id=155150.155154.

[87] Chew Lim Tan JingTao Yao. Guidelines for financial forecasting with neural networks. *International Journal of Forecasting*, 2001. URL http://www.sciencedirect.com/science/article/pii/S0169207097000447.

[88] M.I. Jordan. Serial order: A parallel distributed processing approach. *Tech. Rep*, (8604), 1986. URL http://www.sciencedirect.com/science/article/pii/036402139090002E.

[89] Philippe Jorion. *Value at risk: The new benchmark for controlling market risk*. Irwin Professional Pub. (Chicago). ISBN 0786308486.

[90] Hyun jung Kim and Kyung shik Shin. A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets. *Applied Soft Computing*, 7(2):569 – 576, 2007. ISSN 1568-4946. doi: 10.1016/j.asoc.2006.03.004. URL http://www.sciencedirect.com/science/article/pii/S1568494606000470.

[91] Daniel Kahneman and Amos Tversky. Prospect theory: An analysis of decision under risk. *Econometrica*, 47(2):pp. 263–292, 1979. ISSN 00129682. URL http://www.jstor.org/stable/1914185.

[92] Con Keating and William F Shadwick. An introduction to omega. *Development*, pages 1–15, 2002.

[93] H Kim and K Shin. A hybrid approach based on neural networks and genetic algorithms for detecting temporal patterns in stock markets. *Applied Soft Computing*, 7(2):569–576, Mar 2007. doi: 10.1016/j.asoc.2006.03.004. URL http://linkinghub.elsevier.com/retrieve/pii/S1568494606000470.

[94] Kyoung-jae Kim and Ingoo Han. Genetic algorithms approach to feature discretization in artificial neural networks for the prediction of stock price index. *Expert Systems with Applications*, 19(2):125–132, Aug 2000. doi: 10.1016/S0957-4174(00)00027-0. URL http://linkinghub.elsevier.com/retrieve/pii/S0957417400000270.

[95] Christopher G. Lamoureux and Gary C. Sanger. Firm size and turn-of-the-year effects in the otc/nasdaq market. *The Journal of Finance*, 44(5):pp. 1219–1245, 1989. ISSN 00221082. URL http://www.jstor.org/stable/2328640.

[96] David A Lesmond, Joseph P Ogden, and Charles A Trzcinka. A new estimate of transaction costs. 12(5):1113–1141, 1999.

[97] Baruch Lev and S Ramu Thiagarajan. Fundamental information analysis. *Journal of Accounting Research*, 31(2):190–215, 1993. URL http://www.jstor.org/stable/2491270.

[98] Weimin Li, Jianwei Liu, and Jiajin Le. Using garch-grnn model to forecast financial. *International Journal*, pages 565–574, 2005.

[99] Chang-Chun Lin and Yi-Ting Liu. Genetic algorithms for portfolio selection problems with minimum transaction lots. *European Journal of Operational Research*, 185(1):393–404, Feb 2008. doi: 10.1016/j.ejor.2006.12.024. URL http://linkinghub.elsevier.com/retrieve/pii/S0377221707000057.

[100] Chan Man-chung, Wong Chi-cheong, and L A M Chi-chung. Financial time series forecasting by neural network using conjugate gradient learning algorithm and multiple linear regression weight initialization. *Compute*, 2000.

[101] Harry Markowitz. Portfolio selection. *The Journal of Finance*, 7(1):pp. 77–91, 1952. ISSN 00221082. URL http://www.jstor.org/stable/2975974.

[102] T. Masters. *Advanced algorithms for neural networks: a C++ sourcebook*. Number v. 1. Wiley, 1995. URL http://books.google.no/books?id=R8NQAAAAMAAJ.

[103] Thomas H. McInish, David K. Ding, Chong Soo Pyun, and Udomsak Wongchoti. Short-horizon contrarian and momentum strategies in asian markets: An integrated analysis. *International Review of Financial Analysis*, 17(2): 312 – 329, 2008. ISSN 1057-5219. doi: 10.1016/j.irfa.2006.03.001. URL http://www.sciencedirect.com/science/article/pii/S1057521906000433.

[104] D G McMillan and M E Wohar. Stock return predictability and dividend-price ratio: a nonlinear approach. *International Journal of Finance Economics*, 15(4):351–365, 2010. URL http://onlinelibrary.wiley.com/doi/10.1002/ijfe.401/full.

[105] David G McMillan. Non-linear forecasting of stock returns: Does volume help? *International Journal of Forecasting*, 23(1):115–126, 2007. URL http://www.sciencedirect.com/science/article/B6V92-4M3RNX3-1/2/7fe6b14c497a95ed506414fac3278c88.

[106] Rajnish Mehra and Edward C. Prescott. The equity premium: A puzzle. *Journal of Monetary Economics*, 15(2):145 – 161, 1985. ISSN 0304-3932. doi: 10.1016/0304-3932(85)90061-3. URL http://www.sciencedirect.com/science/article/pii/0304393285900613.

[107] Geoffrey F. Miller, Peter M. Todd, and Shailesh U. Hegde. Designing neural networks using genetic algorithms. In *Proceedings of the third international conference on Genetic algorithms*, pages 379–384, San Francisco, CA, USA, 1989. Morgan Kaufmann Publishers Inc. ISBN 1-55860-006-3. URL http://dl.acm.org/citation.cfm?id=93126.94034.

[108] Vikramjit Mitra, Chia-Jiu Wang, and Satarupa Banerjee. A neuro-svm model for text classification using latent semantic indexing. In *Neural Networks, 2005. IJCNN '05. Proceedings. 2005 IEEE International Joint Conference on*, volume 1, pages 564 – 569 vol. 1, july-4 aug. 2005. doi: 10.1109/IJCNN.2005.1555893.

[109] John Moody and Christian J. Darken. Fast learning in networks of locally-tuned processing units. *Neural Computation*, 1(2):281–294, Jun 1989. doi: 10.1162/neco.1989.1.2.281. URL http://www.mitpressjournals.org/doi/abs/10.1162/neco.1989.1.2.281.

[110] Leung M.T., Chen A.-S., and Daouk H. Forecasting exchange rates using general regression neural networks. *Computers and Operations Research*, 27(11):1093–1110, 2000. doi: doi:10.1016/S0305-0548(99)00144-6. URL http://www.ingentaconnect.com/content/els/03050548/2000/00000027/00000011/art00144.

[111] Mikhail Myagkov and Charles R. Plott. Exchange economies and loss exposure: Experiments exploring prospect theory and competitive equilibria in market environments. *The American Economic Review*, 87(5):pp. 801–828, 1997. ISSN 00028282. URL http://www.jstor.org/stable/2951326.

[112] S. Francis Nicholson. Price ratios in relation to investment results. *Financial Analysts Journal*, 24(1):pp. 105–109, 1968. ISSN 0015198X. URL http://www.jstor.org/stable/4470287.

[113] Jane A Ou and Stephen H Penman. Financial statement analysis and the prediction of stock returns. *Journal of Accounting and Economics*, 11(4):295–329, 1989. URL http://www.sciencedirect.com.library.myebs.de/science/article/B6V87-46KC3N8-2/2/a633e3e82181ef769aeb7013def240ef.

[114] P. Oświe¸cimka, J. Kwapień, and S. Drożdż. Multifractality in the stock market: price increments versus waiting times. *Physica A: Statistical Mechanics and its Applications*, 347(0):626 – 638, 2005. ISSN 0378-4371. doi: 10.1016/j.physa.2004.08.025. URL http://www.sciencedirect.com/science/article/pii/S0378437104011112.

[115] P Pai and W Hong. Forecasting regional electricity load based on recurrent support vector machines with genetic algorithms. *Electric Power Systems Research*, 74(3):417–425, Jun 2005. doi: 10.1016/j.epsr.2005.01.006. URL http://linkinghub.elsevier.com/retrieve/pii/S0378779605000702.

[116] Cheol-Ho Park and Scott H. Irwin. What do we know about the profitability of technical analysis? *Journal of Economic Surveys*, 21(4):786–826, 2007. ISSN 1467-6419. doi: 10.1111/j.1467-6419.2007.00519.x. URL http://dx.doi.org/10.1111/j.1467-6419.2007.00519.x.

[117] John Allen Paulos. *A mathematician plays the stock market*. Perseus Books Group, 2003.

[118] Marc R. and Reinganum. Misspecification of capital asset pricing: Empirical anomalies based on earnings' yields and market values. *Journal of Financial Economics*, 9(1):19 – 46, 1981. ISSN 0304-405X. doi: 10.1016/0304-405X(81)90019-2. URL http://www.sciencedirect.com/science/article/pii/0304405X81900192.

[119] Shivaram Rajgopal, Terry Shevlin, and Mohan Venkatachalam. Does the stock market fully appreciate the implications of leading indicators for future earnings? evidence from order backlog. *Review of Accounting Studies*, 8(4):461–492, 2003. URL http://dx.doi.org/10.1023/A:1027364031775.

[120] M Riedmiller and H Braun. A direct adaptive method for faster backpropagation learning: the rprop algorithm. *IEEE International Conference on Neural Networks*, 1(3):586–591, 1993. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=298623.

[121] S W Roberts. Control chart tests based on geometric moving averages. *Technometrics*, 42(1):239–250, 2000. URL http://www.jstor.org/stable/10.2307/1266443.

[122] F Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. URL http://www.ncbi.nlm.nih.gov/pubmed/13602029.

[123] D E Rumelhart, G E Hinton, and R J Williams. *Learning internal representations by error propagation*, volume 1, pages 318–362. MIT Press, 1986. URL http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA164453.

[124] E.W. Saad, D.V. Prokhorov, and II Wunsch, D.C. Comparative study of stock trend prediction using time delay, recurrent and probabilistic neural networks. *Neural Networks, IEEE Transactions on*, 9(6):1456 –1470, nov 1998. ISSN 1045-9227. doi: 10.1109/72.728395.

[125] Massimo Santini and Andrea Tettamanzi. Genetic programming for financial time series prediction. 2038:361–370, 2001. URL http://dx.doi.org/10.1007/3-540-45355-5_29. 10.1007/3-540-45355-5_29.

[126] N. Sapankevych and R. Sankar. Time series prediction using support vector machines: A survey. *Computational Intelligence Magazine, IEEE*, 4(2):24 –38, may 2009. ISSN 1556-603X. doi: 10.1109/MCI.2009.932254.

[127] Dirk Schiereck, Werner De Bondt, and Martin Weber. Contrarian and momentum strategies in germany. *Financial Analysts Journal*, 55(6):104–116, 1999. URL http://www.cfapubs.org/doi/abs/10.2469/faj.v55.n6.2317.

[128] A. Schroeder. *The Snowball: Warren Buffett and the Business of Life*. Bantam Books. Random House Publishing Group, 2009. ISBN 9780553384611. URL http://books.google.no/books?id=OdpWOjo7EeQC.

[129] J.D. Schwager. *Getting Started in Technical Analysis*. Getting Started In. John Wiley, 1999. ISBN 9780471295426. URL http://books.google.no/books?id=dm6EvSzLYNAC.

[130] William F. Sharpe. A simplified model for portfolio analysis. *Management Science*, 9(2):pp. 277–293, 1963. ISSN 00251909. URL http://www.jstor.org/stable/2627407.

[131] William F. Sharpe. Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance*, 19(3):pp. 425–442, 1964. ISSN 00221082. URL http://www.jstor.org/stable/2977928.

[132] William F. Sharpe. Mutual fund performance. *The Journal of Business*, 39(1):pp. 119–138, 1966. ISSN 00219398. URL http://www.jstor.org/stable/2351741.

[133] Zhiwei Shi and Min Han. Support vector echo-state machine for chaotic time-series prediction. *Neural Networks, IEEE Transactions on*, 18(2):359 –372, march 2007. ISSN 1045-9227. doi: 10.1109/TNN.2006.885113.

[134] Richard W. Sias and Laura T. Starks. The day-of-the-week anomaly: The role of institutional investors. *Financial Analysts Journal*, 51(3):pp. 58–67, 1995. ISSN 0015198X. URL http://www.jstor.org/stable/4479847.

[135] J Sietsma and R Dow. Creating artificial neural networks that generalize. *Neural Networks*, 4(1):67–79, 1991. URL http://dx.doi.org/10.1016/0893-6080(91)90033-2.

[136] D.F. Specht. A general regression neural network. *Neural Networks, IEEE Transactions on*, 2(6):568 –576, nov 1991. ISSN 1045-9227. doi: 10.1109/72.97934.

[137] Francis E H Tay and Lijuan Cao. Application of support vector machines in financial time series forecasting. *Omega*, 29(4):309–317, 2001. URL http://linkinghub.elsevier.com/retrieve/pii/S0305048301000263.

[138] M P Taylor and H Allen. The use of technical analysis in the foreign exchange market. *Journal of International Money and Finance*, 11(3): 304–314, 1992. URL http://linkinghub.elsevier.com/retrieve/pii/0261560692900483.

[139] S.J. Taylor. *Modelling Financial Time Series*. World Scientific, 2008. ISBN 9789812770844. URL http://books.google.no/books?id=QcOZ1gkdA2AC.

[140] Igor V. Tetko, David J. Livingstone, and Alexander I. Luik. Neural network studies. 1. comparison of overfitting and overtraining. *Journal of Chemical Information and Computer Sciences*, 35(5):826–833, 1995. doi: 10.1021/ci00027a006. URL http://pubs.acs.org/doi/abs/10.1021/ci00027a006.

[141] Richard H. Thaler. Amomalies: The january effect. *The Journal of Economic Perspectives*, 1(1):pp. 197–201, 1987. ISSN 08953309. URL http://www.jstor.org/stable/1942958.

[142] U Thissen, R van Brakel, A.P de Weijer, W.J Melssen, and L.M.C Buydens. Using support vector machines for time series prediction. *Chemometrics and Intelligent Laboratory Systems*, 69(1-2):35 – 49, 2003. ISSN 0169-7439. doi: 10.1016/S0169-7439(03)00111-4. URL http://www.sciencedirect.com/science/article/pii/S0169743903001114.

[143] Sunti Tirapat and Aekkachai Nittayagasetwat. An investigation of thai listed firms' financial distress using macro and micro variables. *Multinational Finance Journal*, 1999.

[144] A.P Topchy and O.A Lebedko. Neural network training by means of cooperative evolutionary search. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 389(1-2):240 – 241, 1997. ISSN 0168-9002. doi: 10.1016/S0168-9002(97)00139-3. URL http://www.sciencedirect.com/science/article/pii/S0168900297001393. ¡ce:title¿New Computing Techniques in Physics Research V¡/ce:title¿.

[145] John Von Neumann and Oskar Morgenstern. *Theory of Games and Economic Behavior*. Princeton University Press, 1944. URL http://jmvidal.cse.sc.edu/library/neumann44a.pdf.

[146] Rolf W. and Banz. The relationship between return and market value of common stocks. *Journal of Financial Economics*, 9(1):3 – 18, 1981. ISSN 0304-405X. doi: 10.1016/0304-405X(81)90018-0. URL http://www.sciencedirect.com/science/article/pii/0304405X81900180.

[147] Steven Walczak. An empirical analysis of data requirements for financial forecasting with neural networks. *Journal of Management Information Systems*, 17(4):203–222, 2001. URL http://portal.acm.org/citation.cfm?id=1289668.1289677.

[148] R. Weatherford. *Philosophical foundations of probability theory*. International library of philosophy. Routledge & K. Paul, 1982. ISBN 9780710090027. URL http://books.google.no/books?id=B809AAAAIAAJ.

[149] Werner and Kinnebrock. Accelerating the standard backpropagation method using a genetic approach. *Neurocomputing*, 6(5-6):583 – 588, 1994. ISSN 0925-2312. doi: 10.1016/0925-2312(94)90008-6. URL http://www.sciencedirect.com/science/article/pii/0925231294900086.

[150] Ronald J. Williams and David Zipser. A learning algorithm for continually running fully recurrent neural networks. *Neural Comput.*, 1:270–280, June 1989. ISSN 0899-7667. doi: 10.1162/neco.1989.1.2.270. URL `http://dl.acm.org/citation.cfm?id=1351124.1351135`.

[151] Michael Wooldridge. *An Introduction to Multiagent Systems*. Wiley, Chichester, UK, 2. edition, 2009. ISBN 978-0-470-51946-2.

[152] Junyan Yang and Youyun Zhang. Application research of support vector machines. *Science*, pages 857–864, 2005.

[153] Xin Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87 (9):1423 –1447, sep 1999. ISSN 0018-9219. doi: 10.1109/5.784219.

[154] G Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003. URL `http://linkinghub.elsevier.com/retrieve/pii/S0925231201007020`.

[155] G.P. Zhang. Avoiding pitfalls in neural network research. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 37(1):3 –16, jan. 2007. ISSN 1094-6977. doi: 10.1109/TSMCC.2006.876059.

[156] Guoqiang Zhang, B. Eddy Patuwo, and Michael Y. Hu. Forecasting with artificial neural networks:: The state of the art. *International Journal of Forecasting*, 14(1):35 – 62, 1998. ISSN 0169-2070. doi: 10.1016/S0169-2070(97)00044-7. URL `http://www.sciencedirect.com/science/article/pii/S0169207097000447`.

[157] Ping Zhang, Y. Sankai, and M. Ohta. Hybrid adaptive learning control of nonlinear system. In *American Control Conference, 1995. Proceedings of the*, volume 4, pages 2744 –2748 vol.4, jun 1995. doi: 10.1109/ACC.1995.532348.

[158] Wei Zhang, Qing Cao, and Marc J Schniederjans. Neural network earnings per share forecasting models: A comparative analysis of alternative methods. *Decision Sciences*, 35(2):205–237, 2004. URL `http://doi.wiley.com/10.1111/j.00117315.2004.02674.x`.

[159] Yanglan Zhang and Yu Hua. Portfolio optimization for multi-stage capital. *Portfolio The Magazine Of The Fine Arts*, pages 982–987, 2004.

Index