EVOLVING BIOLOGICALLY INSPIRED TRADING ALGORITHMS

Master Thesis in Informatics

Patrick Gabrielsson

2012MAGI06

HÖGSKOLAN I BORÅS
INSTITUTIONEN FÖR DATA- OCH AFFÄRSVETENSKAP

Svensk titel: Evolvering av Biologiskt Inspirerade Handelsalgoritmer

Engelsk titel: Evolving Biologically Inspired Trading Algorithms

Utgivningsår: 2012

Författare: Patrick Gabrielsson

Handledare: Rikard König

Keywords: Algorithmic Trading, Hierarchical Temporal Memory, Neural Networks, Machine Learning, Predictive Modeling, Classification, Evolutionary Computing, Genetic Algorithm, Optimization

Abstract

One group of information systems that have attracted a lot of attention during the past decade are financial information systems, especially systems pertaining to financial markets and electronic trading. Delivering accurate and timely information to traders substantially increases their chances of making better trading decisions.

Since the dawn of electronic exchanges the trading community has seen a proliferation of computer-based intelligence within the field, enabled by an exponential growth of processing power and storage capacity due to advancements in computer technology. The financial benefits associated with outperforming the market and gaining leverage over the competition has fueled the research of computational intelligence in financial information systems. This has resulted in a plethora of different techniques.

The most prevalent techniques used within algorithmic trading today consist of various machine learning technologies, borrowed from the field of data mining. Neural networks have shown exceptional predictive capabilities time and time again.

One recent machine learning technology that has shown great potential is Hierarchical Temporal Memory (HTM). It borrows concepts from neural networks, Bayesian networks and makes use of spatiotemporal clustering techniques to handle noisy inputs and to create invariant representations of patterns discovered in its input stream. In a previous paper [1], an initial study was carried-out where the predictive performance of the HTM technology was investigated within algorithmic trading of financial markets. The study showed promising results, in which the HTM-based algorithm was profitable across bullish-, bearish and horizontal market trends, yielding comparable results to its neural network benchmark. Although, the previous work lacked any attempt to produce near optimal trading models.

Evolutionary optimization methods are commonly regarded as superior to alternative methods. The simplest evolutionary optimization technique is the genetic algorithm, which is based on Charles Darwin's evolutionary theory of natural selection and survival of the fittest. The genetic algorithm combines exploration and exploitation in the search for optimal models in the solution space.

This paper extends the HTM-based trading algorithm, developed in the previous work, by employing the genetic algorithm as an optimization method. Once again, neural networks are used as the benchmark technology since they are by far the most prevalent modeling technique used for predicting financial markets. Predictive models were trained, validated and tested using feature vectors consisting of technical indicators, derived from the E-mini S&P 500 index futures market.

The results show that the genetic algorithm succeeded in finding predictive models with good performance and generalization ability. The HTM models outperformed the neural network models, but both technologies yielded profitable results with above average accuracy.

Contents

PART I Introduction ..... - 1 -
1 Introduction ..... - 2 -
1.1 Problem Statement and Research Questions ..... - 4 -
1.2 Purpose ..... - 5 -
1.3 Scope ..... - 5 -
1.4 Research Objectives ..... - 5 -
1.5 Main Contributions ..... - 5 -
1.6 Thesis Outline ..... - 6 -
2 Research Method ..... - 7 -
2.1 Research Strategy ..... - 7 -
2.2 Research Design ..... - 8 -
PART II Theory ..... - 10 -
3 The Nature of Financial Markets ..... - 11 -
3.1 The Market Mechanism ..... - 11 -
3.2 The Law of Supply and Demand ..... - 12 -
3.3 The Efficient Market Hypothesis ..... - 13 -
4 Financial Market Analysis ..... - 14 -
4.1 Fundamental Analysis ..... - 14 -
4.2 Technical Analysis ..... - 15 -
4.3 Random Walk Theory ..... - 18 -
4.4 Computational Intelligence in Financial Engineering ..... - 18 -
5 Trading Financial Markets ..... - 19 -
5.1 Assets ..... - 19 -
5.2 Indices ..... - 19 -
5.3 Derivatives ..... - 19 -
5.4 Contracts ..... - 20 -
5.5 Over-The-Counter (OTC) Markets ..... - 20 -
5.6 Exchange-Traded Markets ..... - 21 -
5.7 Margins, Leverage and Fees ..... - 21 -
5.8 Electronic Exchanges ..... - 22 -
5.9 Trading Styles ..... - 22 -
5.10 Order Types ..... - 24 -
6 The Data Mining Process ..... - 25 -
6.1 Business Understanding ..... - 28 -
6.2 Data Understanding ..... - 29 -
6.2.1 Collecting Initial Data ..... - 29 -
6.2.2 Describing the Data ..... - 29 -
6.2.3 Exploring the Data ..... - 30 -
6.2.4 Verifying Data Quality ..... - 30 -
6.3 Data Preparation ..... - 31 -
6.3.1 Selecting the Data ..... - 31 -
6.3.2 Cleaning the Data ..... - 31 -
6.3.3 Constructing the Data ..... - 33 -
6.3.4 Integrating the Data ..... - 35 -
6.3.5 Formatting the Data ..... - 35 -
6.4 Modeling ..... - 37 -
6.4.1 Selecting the Modeling Technique ..... - 37 -
6.4.2 Generating the Test Design ..... - 39 -

6.4.3 Building the Models ..... - 42 -
6.4.4 Assessing the Models ..... - 42 -
6.5 Evaluation ..... - 42 -
6.6 Deployment ..... - 42 -
7 Neural Networks ..... - 42 -
7.1 The Biological System ..... - 42 -
7.2 The Perceptron ..... - 43 -
7.3 The Multi-Layer Perceptron (MLP) ..... - 46 -
8 Hierarchical Temporal Memory (HTM) ..... - 50 -
8.1 The Biological System ..... - 51 -
8.2 Memory Prediction Theory of Brain Function ..... - 52 -
8.3 A Mathematical Model of Brain Function ..... - 53 -
8.4 HTM Concepts and Terminology ..... - 53 -
8.5 The HTM Learning Algorithms ..... - 57 -
9 Evolutionary Learning ..... - 67 -
9.1 The Biological System ..... - 68 -
9.2 The Genetic Algorithm ..... - 69 -
10 Related Work ..... - 73 -
10.1 Neural Networks and Evolutionary Learning ..... - 73 -
10.2 Hierarchical Temporal Memory ..... - 74 -
PART III Experimental Design and Execution ..... - 76 -
11 Hypotheses ..... - 77 -
11.1 Hypothesis 1 ..... - 77 -
11.2 Hypothesis 2 ..... - 79 -
11.3 Hypothesis 3 ..... - 80 -
11.4 Hypothesis 4 ..... - 81 -
12 Method ..... - 81 -
12.1 Data Acquisition ..... - 82 -
12.2 Aggregation ..... - 83 -
12.3 Feature Extraction ..... - 85 -
12.4 Dataset Partitioning ..... - 85 -
12.5 Class Definition and Trading Strategy ..... - 88 -
12.6 Performance Measure ..... - 88 -
12.7 Model Optimization ..... - 89 -
13 Experiments ..... - 89 -
13.1 Evolving the HTM Models ..... - 89 -
13.2 Evolving the ANN Models ..... - 96 -
13.3 Evaluating Model Performance ..... - 98 -
PART IV Results and Conclusions ..... - 99 -
14 Results ..... - 100 -
14.1 HTM Optimization Results ..... - 101 -
14.2 ANN Optimization Results ..... - 105 -
14.3 HTM Versus ANN Performance Comparison ..... - 107 -
15 Discussion ..... - 108 -
15.1 Comparison to theory and hypotheses ..... - 108 -
15.2 Comparison to previous work ..... - 110 -
15.3 Reliability, Replicability and Validity ..... - 111 -
16 Conclusions and Future Work ..... - 112 -
16.1 Conclusions ..... - 112 -
16.2 Future Work ..... - 113 -
References ..... - 115 -

Appendix A ..... - 121 -
Simple Moving Average (SMA) ..... - 121 -
Exponential Moving Average (EMA) ..... - 122 -
Percentage Price Oscillator (PPO) ..... - 123 -
PPO Signal Line (PPO_{EMA}) ..... - 124 -
PPO Histogram (PPO_{HIST}) ..... - 124 -
Relative Strength Index (RSI) ..... - 125 -
William's %R (Wm%R) ..... - 127 -
Normalized Volatility Indicator (NVI) ..... - 128 -
Chaikin Money Flow (CMF) ..... - 130 -
Bollinger Bands %B (%B) ..... - 131 -
Rate of Change (ROC) ..... - 133 -
Fast Stochastic Oscillator (%K) ..... - 134 -
Fast Stochastic Oscillator Signal Line – %D(3) ..... - 134 -

List of Figures

Figure 3.1 The supply and demand curve. - 12 -
Figure 5.1 Order Book. - 24 -
Figure 6.1 The multidisciplinary nature of Data Mining. - 26 -
Figure 6.2 The Knowledge Discovery in Databases (KDD) process. - 26 -
Figure 6.3 The Cross Industry Standard for Data Mining (CRISP-DM) process. - 27 -
Figure 6.4 Time series with noise (top) and without noise (bottom). - 32 -
Figure 6.5 Time series with artifacts (red) and with artifacts eliminated (blue). - 32 -
Figure 6.6 Decision boundary for a 2-class problem with 3 independent attributes. - 38 -
Figure 6.7 Confusion matrix for a two class problem. - 40 -
Figure 6.8 ROC curve for a good classifier (blue) and for a random classifier (red). - 41 -
Figure 7.1 The Perceptron - 43 -
Figure 7.2 The Single-Layer Perceptron Network - 44 -
Figure 7.3 The OR Perceptron - 45 -
Figure 7.4 Input space for the OR problem (left) and the XOR problem (right). - 46 -
Figure 7.5 The XOR MLP Network - 46 -
Figure 8.1 The Microcolumn structure in the cerebral cortex. - 51 -
Figure 8.2 A hierarchical network of microcolumns in the cerebral cortex. - 52 -
Figure 8.3 Discovering causes of patterns in the world. - 54 -
Figure 8.4 Structure of an HTM Network. - 55 -
Figure 8.5 Spatial Patterns in a Binary Image. - 55 -
Figure 8.6 Temporal Patterns in a Binary Image. - 56 -
Figure 8.7 Three layer HTM network with a 32x32 pixel input image. - 57 -
Figure 8.8 HTM node with spatial and temporal pooler. - 58 -
Figure 8.9 The learning stages of an HTM node. - 60 -
Figure 8.10 An HTM node's maxDistance setting for spatial clustering. - 60 -
Figure 8.11 Time-adjacency matrix. - 62 -
Figure 8.12 Greedy algorithm for forming temporal groups (topNeighbors = 2). - 63 -
Figure 8.13 HTM node's inference procedure for three consecutive patterns. - 65 -
Figure 8.14 Inference/learning procedure for nodes in a hierarchy. - 66 -
Figure 8.15 HTM network configured as a classifier. - 67 -
Figure 9.1 Chromosome representation of the restaurant problem. - 70 -
Figure 9.2 Two generations of the chromosome population. - 71 -
Figure 9.3 Crossover operations. - 72 -
Figure 9.4 Mutation operator. - 72 -
Figure 12.1 Barplot of the E-mini S&P 500 futures index market - 83 -
Figure 12.2 Candle stick plot of the E-mini S&P 500 futures index market - 84 -
Figure 12.3 Volume plot of the E-mini S&P 500 futures index market - 84 -
Figure 12.4 The dataset for the E-mini S&P 500 - 87 -
Figure 14.1 Class Distribution Per Dataset - 100 -
Figure A.1 Closing price with two SMAs for the E-mini S&P 500 - 121 -
Figure A.2 Closing price with two EMAs for the E-mini S&P 500 - 122 -
Figure A.3 PPO, its signal line and histogram for the E-mini S&P 500 - 123 -
Figure A.4 RSI for the E-mini S&P 500 - 126 -
Figure A.5 Wm%R for the E-mini S&P 500 - 127 -

Figure A.6 NVI for the E-mini S&P 500..... - 129 -
Figure A.7 CMF (top) and volume (bottom) for the E-mini S&P 500 ..... - 130 -
Figure A.8 %B (bottom) and BB bands (top) for the E-mini S&P 500 ..... - 132 -
Figure A.9 ROC for the E-mini S&P 500 ..... - 133 -
Figure A.10 %K and %D for the E-mini S&P 500 ..... - 134 -

List of Tables

Table 6.1 Data Mining Process Mapping ..... - 27 -
Table 6.2 CRISP-DM Phases ..... - 28 -
Table 7.1 OR Truth Table ..... - 45 -
Table 12.1 SlickCharts File Format ..... - 82 -
Table 12.2 Technical Indicators ..... - 85 -
Table 12.3 Class Definitions ..... - 88 -
Table 12.4 Trading Strategy ..... - 88 -
Table 13.1 HTM Parameters ..... - 90 -
Table 13.2 HTM Network Topologies (topmost classifier node not included) ..... - 92 -
Table 13.3 HTM Chromosome ..... - 95 -
Table 13.4 HTM Genetic Algorithm Settings ..... - 96 -
Table 13.5 ANN Chromosome ..... - 97 -
Table 13.6 ANN Parameters ..... - 97 -
Table 13.7 ANN Genetic Algorithm Settings ..... - 98 -
Table 14.1 Class Distribution Per Dataset ..... - 100 -
Table 14.2 HTM Optimization Results ..... - 102 -
Table 14.3 HTM Performance ..... - 103 -
Table 14.4 HTM Average Performance ..... - 104 -
Table 14.5 ANN Optimization Results ..... - 105 -
Table 14.6 ANN Performance ..... - 106 -
Table 14.7 ANN Average Performance ..... - 107 -

Acknowledgements

I would like to thank Ulf Johansson and Rikard König at the School of Business and Information Technology at the University of Borås in Sweden for sharing their data mining knowledge with me prior to and during this thesis work. A special thanks to my supervisor Rikard König for his useful comments an support.

Borås, May 2012 Patrick Gabrielsson

PART I
Introduction

1 Introduction

"*Informatics is the science of information, the practice of information processing, and the engineering of information systems. Informatics studies the structure, algorithms, behavior, and interactions of natural and artificial systems that store, process, access and communicate information ... Loosely, it can be thought of as studying how to design a system that delivers the right information, to the right person in the right place and time, in the right way* " [2]. One group of information systems that have attracted a lot of attention during the past decade are financial information systems, especially systems pertaining to financial markets and electronic trading. Delivering accurate and timely information to traders substantially increases their chances of making better trading decisions.

Trading as a profession has been around for centuries, from ancient merchants exchanging goods with each other using primitive bartering systems, to the introduction of monetary systems and contemporary electronic trading systems. Two decades ago, trades were negotiated by *pit-traders* in something called the *open-outcry* market. Traders in colorful jackets would gather in an area in the middle of the trading floor that looked like an open pit. When a client called in an order, it was received by a *broker*, who scribbled down the order details on a piece of paper and handed it over to a *runner*. The runner would literally run between the broker and the trading pit, where he would hand over the paper slip to a trader. Using a special kind of sign language and a loud voice, the trader would then negotiate a trade with another trader. Appropriately, this type of trading environment was called the pit-traded open-outcry. Although some pit-traded markets still exist today, most of them were replaced by electronic exchanges over a decade ago, where the investors, acting as traders, enter their own orders into the market through a graphical user interface presented to them on their computer screens. The order is routed to the electronic exchange where it eventually trades with a counterparty's order. This contraption enabled computer-mediated competition of human traders in the market place. Soon the trading community realized that considerable leverage could be gained over the competition by replacing human traders with computer-based traders that could process vast amounts of information in fractions of a second.

Since the dawn of electronic exchanges the trading community has seen a proliferation of computer-based intelligence within the field, enabled by an exponential growth of processing power and storage capacity due to advancements in computer technology. The financial benefits associated with outperforming the market and gaining leverage over the competition has fueled the research of computational intelligence in financial information systems. This has resulted in a plethora of different techniques. The most famous case was based on agent theory, from within the field of artificial intelligence, in which a group of IBM researchers proved that computer-based, autonomous trading agents could outperform their human counterparts by using a strategic bidding algorithm [3]. A refined version of their original algorithm once again showed its superiority over human traders ten years later [4].

In 1970, Eugene Fama released a paper entitled *"Efficient Capital Markets: A Review of Theory and Empirical Work"* [5], in which he proposed that financial markets are

efficient to such a degree that any opportunity that arises in the market would be assimilated before it could be exploited. He also suggested that the future direction of the market cannot be predicted using any past information. This caused quite a stirrup in the community since his proposal essentially implied that a competitive advantage could not be gained by projecting historical information into the future. Even though Fama has not yet been proven wrong, in theory, multiple empirical studies have shown that, in practice, markets aren't quite as efficient as Fama hypothesized. In fact, large financial institutions realize annual profits into the millions by basing their decisions on past market movements and fundamental factors. The reason for the discrepancy between Fama's efficient market hypothesis and the real world is attributed to the imbalance of public information between market participants, where more privileged investors gain considerable leverage from inside information. Furthermore, trading psychology plays a major role, where market participants react differently to financial news events, and hence financial markets do not follow a rational process in practice. Financial markets also demonstrate seasonal recurrences and business cycles.

Some modeling techniques assume that financial markets follow an ordered, random walk, more specifically, they employ chaos theory. Chaos theory states that even in randomness there exists order. A lot of processes found in nature may seem random but in fact there exists order in that randomness, for example the temperature fluctuations at a certain geographical location may seem random, but they do show deterministic, reoccurring seasonal changes. Stock prices are often modeled as continuous-time stochastic processes, in which it is assumed that each stochastic variable is *Markov* and follows a normal distribution. One such well-known stochastic process in the *Wiener process*, a type of *Markov stochastic process*, in which each stochastic variable follows a standard normal distribution. The Wiener process has is also used in physics to describe the motion of a particle subjected to a large number of collisions with other molecular particles, where it is referred to as *Brownian motion*. A further enhancement of the Wiener process is the *Itô process*.

Other approaches use linear regression models from statistics to extrapolate future values, such as univariate and multivariate linear discriminant analysis. Even though such techniques have shown good predictive capabilities for short time horizons, studies have shown that financial markets cannot be modeled entirely using linear relationships. Thus, combinations of linear techniques and chaos theory have yielded better results.

The most prevalent techniques used within algorithmic trading today consist of various machine learning technologies, borrowed from the field of data mining. Neural networks have shown exceptional predictive capabilities time and time again. Although, neural networks produce opaque (non-transparent) models, which constitutes a problem in some application areas where regulation demands that decisions are traceable and explainable in detail. By sacrificing some degree of accuracy for transparency, decision trees and other rule-generating techniques are the natural remedy to such constraints. Probabilistic techniques, such as Bayesian networks and Fuzzy networks, have also gained popularity by incorporating the ability to deal with uncertainty in predictive modeling of financial markets.

One recent machine learning technology that has shown great potential is Hierarchical Temporal Memory (HTM). It borrows concepts from neural networks, Bayesian networks and makes use of spatiotemporal clustering techniques to handle noisy inputs and to create invariant representations of patterns discovered in its input stream. In a previous paper, an initial study was carried-out where the predictive performance of the HTM technology was investigated within algorithmic trading of financial markets [1]. The study showed promising results, in which the HTM-based algorithm was profitable across bullish-, bearish and horizontal market trends, yielding comparable results to its neural network benchmark. Although, the previous work lacked any attempt to produce near optimal trading models.

Evolutionary optimization methods are commonly regarded as superior to alternative methods. The simplest evolutionary optimization technique is the genetic algorithm, which is based on Charles Darwin's evolutionary theory of natural selection and survival of the fittest. The genetic algorithm combines exploration and exploitation in the search for optimal models in the solution space.

1.1 Problem Statement and Research Questions

The introductory text emphasizes the importance of computational intelligence in financial information systems, specifically trading systems, where computer-based trading algorithms tend to outperform their human counterparts. This fact is unequivocally true in high-frequency trading scenarios where trading decisions need to be made within fractions of a second. Furthermore, the algorithmic trading community continuously investigates novel techniques for designing trading algorithms capable of outperforming the competition, and hence, yielding substantial financial returns for their beneficiaries. Hence, research within the field of computer-based intelligence in financial engineering and economics is highly relevant.

The introductory text also emphasizes the great potential of biologically-inspired learning models in discovering and exploiting patterns in financial time series. Artificial neural networks have a proven track record throughout the literature. Furthermore, evolutionary optimization methods, based on Charles Darwin's evolutionary theory of natural selection and survival of the fittest, have been combined with artificial neural networks to procedure highly accurate predictive models.

Hierarchical Temporal Memory (HTM) is a relatively novel biologically-inspired model, which was initially investigated within algorithmic trading of financial markets in [1] and benchmarked against artificial neural networks.

This paper extends the HTM-based trading algorithm, developed in the previous work, by employing the genetic algorithm as an optimization method. Once again, neural networks are used as the benchmark technology since they are by far the most prevalent modeling technique used for predicting financial markets.

This provides the motivation for the study underlying this thesis and raises the following four research questions:

1. Is it possible to implement profitable, predictive HTM-based classification models for financial time series? (RQ.1)
2. Can HTM-based models be optimized using evolutionary learning? (RQ.2)
3. How well do HTM-based models generalize to novel data? (RQ.3)
4. How does the performance of HTM-based models compare to artificial neural networks? (RQ.4)

1.2 Purpose

Based on the introductory text and the problem statement, the aim of this thesis is to evaluate the suitability of HTM-based, algorithmic trading models, optimized using an evolutionary technique and to compare model performance against a benchmark technology in the form of recurrent neural networks. The goal is to design a profitable HTM-based trading model yielding consistent returns though multiple iterations over multiple financial datasets.

1.3 Scope

This paper focuses on a subset of the plentitude of biologically-inspired models; namely artificial neural networks, hierarchical temporal memory and the genetic algorithm. The financial data used to evaluate model performance is limited to the E-mini S&P 500 index futures market, although multiple runs over multiple datasets are carried-out using a modified cross validation technique in order to yield a sufficient estimate of model generalization ability. This scope is deemed necessary and sufficient in order to support the purpose of the study.

1.4 Research Objectives

A direct translation of the research questions, defined in the problem statement, results in the following four main research objectives, as outlined below:

1. Implement profitable, predictive HTM-based classification models for financial time series. (RO.1)
2. Optimize and evaluate model performance using evolutionary learning. (RO.2)
3. Evaluate model generalization ability using a modified cross-validation technique. (RO.3)
4. Benchmark HTM technology performance against recurrent neural networks. (RO.4)

1.5 Main Contributions

The novelty in this paper consists of combining the HTM technology with evolutionary optimization of predictive classification models in order to produce timely and accurate signals for trading financial markets. To the best of the author's knowledge, such a study has not been performed before and therefore constitutes the main contribution to the research community.

The performance of HTM-based trading models, optimized with the genetic algorithm, are compared to optimized recurrent neural networks, which also provides valuable insight into the performance between both technologies. This paper also uses a modified cross-validation scheme for dealing with sequential data, although similar techniques have been employed before.

1.6 Thesis Outline

This thesis is organized into 16 chapters (including this chapter) a references chapter and one appendix.

This chapter has introduced the topic for this theses, defined the research questions and research objectives, stated the purpose and scope of the thesis and expressed the main contributions provided by the thesis. The following chapter, *Chapter 2*, *Research Method*, sets up and discusses the research strategy and research design employed for this thesis work.

First of all, it is imperative to understand the problem domain for which a solution is sought. The problem domain in this paper is financial markets, more specifically, trading financial markets. *Chapter 3, The Nature of Financial Markets*, introduces basic financial and economic concepts. *Chapter 4, Financial Market Analysis* presents common techniques for analyzing financial markets. *Chapter 5, Trading Financial Markets*, defines some common terminology and concepts related to trading financial markets.

Once the problem domain has been introduced, *Chapter 6, The Data Mining Process*, gives a general overview of the process involved in extracting useful information from data, focusing on predictive modeling. Common preprocessing, training, validation and testing methods are explained including various performance measures used to evaluate models.

The next three chapters give detailed descriptions of the three specific machine learning technologies used in this paper for predictive modeling of financial markets. Each chapter includes an introductory subchapter on the biological system from which the technology finds its inspiration. *Chapter 7, Neural Networks*, describes neural networks, the machine learning technology based on interconnecting neurons in the mammalian nervous system. *Chapter 8, Hierarchical Temporal Memory*, takes the concept of communicating neurons to a higher level of abstraction. This relatively new machine learning technology is based on groups of neurons, organized into vertical columns across the six layers in the neocortex of the mammalian brain, acting together as single processing units. *Chapter 9, Evolutionary Learning*, details the genetic algorithm, which draws its inspiration from genetics and Darwin's evolutionary theory of natural selection and survival of the fittest.

Chapter 10, Related Work, discusses previous work done in the area of predictive modeling of financial markets. Neural networks and evolutionary learning have been around for a relatively long time, which is evident from the abundance of studies available using these two approaches. When it comes to the relatively novel Hierarchical Temporal Memory technology, few studies have been carried out with regards to its

application within predictive modeling of financial markets. Therefore, the related work for the HTM technology includes studies from various application areas.

Chapter 11, *Hypotheses*, elaborates on the theory presented in *Chapters 3-10* in order to produce a number of hypotheses supporting the research questions and the research objectives. The results from the experiments, in *Chapter 14*, were then used to prove the hypotheses right or wrong.

Chapter 12, Method, Chapter 13, *Experiments* and *Chapter 14, Results*, describe the method used to produce the predictive models in this paper, the experiments carried out and their associated results, respectively. This is followed by a discussion in *Chapter 15, Discussion*. Finally, conclusive remarks and an elaboration on suggested future work are presented in *Chapter 16, Conclusions and Future Work*.

All references used in this paper are listed in the *References* chapter. *Appendix A* contains descriptions, plots and mathematical formulas for various technical indicators.

The various chapters have been grouped under four parts. *Part 1, Introduction*, contains *Chapter 1*, *Introduction* and *Chapter 2*, *Research Method*.

Part II, Theory consists of Chapters 3-10, *The Nature of Financial Markets*, *Financial Market Analysis*, *Trading Financial Markets*, *The Data Mining Process*, *Neural Networks*, *Hierarchical Temporal Memory*, *Evolutionary Learning* and *Related Work*.

Part III, *Experimental Design and Execution*, includes *Chapter 11*, *Hypotheses*, *Chapter 12*, *Method* and *Chapter 13*, *Experiments*.

Part IV, *Results and Conclusions*, contains *Chapter 14*, *Results*, *Chapter 15*, *Discussion* and *Chapter 16*, *Conclusions and Future Work*.

2 Research Method

This chapter describes the research method employed in this thesis and discusses the suitability of the chosen approach with regards to the nature of the problem outlined in the previous chapter.

2.1 Research Strategy

The relationship between research and theory comes in two main flavors; deductive theory and inductive theory.

Using a deductive approach, theory constitutes the entry point into the research process. In other words, the research process begins with a literature study of a specific domain from which ideas can emerge or from which problems can be identified. The literature study results in the formulation of hypotheses in the form of propositional implications that can be proven to be true of false. The next step in the deductive approach is to identify and acquire suitable data for the investigation. Through observation, the investigation results in a number of findings. The findings are then used to evaluate the

hypotheses. The insights obtained from the outcome of the hypotheses evaluation are finally used to update the theory, hence completing the research cycle [6].

Using an inductive approach, the relationship between theory and research is reversed. In other words, the process starts with acquiring data through observations. The data is then analyzed and scrutinized resulting in a number of findings. Finally, the findings are used to update the theory [6].

The nature of the problem at hand in this thesis called for the employment of a deductive approach. The theory behind financial markets, specifically market micro-structure, and various methods for analyzing them needed to be understood. Furthermore, the theory of biologically-inspired predictive models and evolutionary optimization methods needed to be researched. It was also desired to evaluate the optimized predictive models by exposing them to financial times series - a purely quantitative strategy. Finally, it was deemed beneficial to use the Cross Industry Standard Process for Data Mining (CRISP-DM) while conducting the study (chapter 6).

From an epistemological point of view, the scientific method was adopted. The scientific method is based on positivism, a natural science epistemology, which advocates the importance of adhering to the methods of the natural sciences when conducting research. This means that only phenomena confirmed by the senses is deemed as knowledge, that the researcher adopts a high degree of objectivity during the research and that facts are arrived at by collecting data to prove prefabricated hypotheses right of wrong [6]. The opposite viewpoint, interpretism, which is highly subjective and open to a high degree of interpretation by the researcher was deemed inappropriate for the purely quantitative research strategy employed in this thesis.

When it comes to the philosophical standpoint with regards to ontology, the scientific method assumes an objectivistic position, i.e. that the objects in the world exists regardless of the influence of social actors. In other words, objects in the world do not exists because of a causal relationship with social actors, an ontology referred to as constructionalism [6]. Even though social, psychological and political factors are known to influence financial markets, the research conducted in this thesis assumes objectivism. The rationale behind this assumption is made clear in the discussion of technical analysis in chapter 4.

2.2 Research Design

As a consequence of adopting a positivistic, objectivistic and purely quantitative research strategy, an experimental research design was a natural choice.

In classical experimental design, two groups are formed; one constituting the group under investigation and the other a control group. Usually, the main focus of attention is determining causual relationships between independent variables and a dependent variable [6]. As an example, in pharmaceutical research involving a new drug, one might want to determine if the new drug can elevate some physiological condition. By dividing a population of test subjects into two groups, the test group and the control group, and

only administering the drug (independent variable) to the test group, a comparison with respect to the effects of the drug on the physiological condition (dependent variable) can be evaluated. Most commonly, the test subjects in the control group are administered a placebo drug, i.e. a drug with no effect, so as to rule out any psychological factors.

As a direct analogy, the experimental design employed in this research uses HTM-based models (chapter 8) as the test subjects in the test group and recurrent neural networks (chapter 7) in the control group. Although, in this case, the HTM-based models and the neural networks are exposed to the same stimulus, i.e. financial indicators (independent variables) while the profit they make (dependent variable), based on their trading decisions, is evaluated. The neural networks are used as a benchmark technology in the experimental design. A detailed description of the experimental design is given in chapters 12 and 13. The results from the experiments were then analyzed and evaluated quantitatively (chapter 14).

Other important considerations are the reliability, replicability and validity of the study.

Reliability has to do with the stability of measurements used during the study [6]. This thesis employs technical indicators, which are standardized measures derived from the financial time series and therefore inherently constitute a high degree of reliability.

Replicability is concerned with the reproducibility of the experiments [6]. The method used during data acquisition, preprocessing, feature extraction, modeling and analysis is meticulously described in following chapters 12 and 13, hence yielding a high degree of replicability.

Finally, the validity of the study can be divided into three parts; measurement validity, internal validity and external validity.

Measurement validity relates to if a derived measure really reflects the concept it is supposedly measuring [6]. As with reliability, the same argument can be used to suggest a high degree of measurement validity, i.e. since the technical indicators, used as the derived measures of characteristics of financial markets, are standardized measures, they also imply a high degree of measurement validity.

The internal validity is concerned with the causality between the independent variables and the dependent variable, i.e. if the independent variables really account for the variation in the dependent variable [6].

The external validity is a direct assessment of the generalization ability of the study [6].

The study's reliability, replicability and validity are discussed in *Chapter 15*, *Discussion*.

PART II
Theory

3 The Nature of Financial Markets

This section describes three basic concepts relating to financial markets; the market mechanism, the law of supply and demand and the efficient market hypothesis.

3.1 The Market Mechanism

When two or more parties come together to trade goods with each other they are required to follow a specific protocol known as a *market mechanism*. The market mechanism governs the rules and procedures for all market participants acting within a market. One specific type of market mechanism is the *auction*. The two most well known auctions are the *English Auction* and the *Dutch Flower Auction*. In an English auction, one seller's goods are auctioned out to a pool of buyers. Once the auctioneer has announced an initial price for the seller's goods, the buyers engage in a competitive bidding procedure, subsequently outbidding each other until a final bid has been reached. The seller's goods are then delivered to the buyer in exchange for a monetary equivalent of the buyer's final bid. In a Dutch flower auction, the same bidding procedure is followed, but the roles of the buyers and sellers are reversed, i.e. multiple sellers, selling the same type of goods, complete to sell their goods to a single buyer. Common to both types of auctions, is that they are one-sided, sequential and centralized. The one-sidedness is given in the English auction by multiple buyers bidding for one seller's goods, and in the Dutch flower auction multiple sellers bid for one buyer's money. The bidding process is sequential, since market participants outbid each other one after the other. The two auction mechanisms are centralized in the sense that the auctioneer has complete and perfect knowledge of all the participants' preferences, i.e. all the market participant's intentions are publicly and centrally known.

The most common market mechanism employed in electronic, financial markets is the *Continuous Double Auction (CDA)*. The CDA borrows and combines parts from both the English auction and the Dutch flower auction to create a market mechanism containing multiple buyers and sellers. This is what is meant by a double auction, where both buyers and sellers can bid for (negotiate) a trade. Furthermore, the trading process is continuous, where buyers' and sellers' bids are continuously matched and cleared, i.e. as soon as a buyer and seller agree upon a price a trade is executed. Usually, the market mechanism for electronic, financial markets is defined as a *decentralized, asynchronous CDA*. Decentralized, since the public information is not centrally known, i.e. there is no central agent (auctioneer) with complete and perfect knowledge about all market participant's intentions. The CDA is asynchronous, since the buyers and sellers can place bids into the market at any point in time.

For electronic, financial markets, the *market protocol* also defines a set of interaction-, clearing- and pricing rules. *Interaction rules* define the set of allowable actions for market participants, including the format of submitted bids. *Clearing rules* define the exact circumstances under which a transaction occurs. *Pricing rules* define, among other things, the price of the transaction.

The imbalance of buyers' demand for a certain product and the sellers' supply of that specific product gives rise to a basic economic law known as *the law of supply and demand* which is described in the next section.

3.2 The Law of Supply and Demand

One of the most fundamental concepts in economics is *the law of supply and demand*. It defines a correlation between the supply and demand of a product with the price of that product. If the seller' supply of a certain product is in equilibrium with buyers' demand of that specific product, the market price of the product remains stable. If, on the other hand, there is an imbalance between the supply and demand of a product, it drives the market price of that product in a certain direction. If there is an abundance in the supply of the product in the market and little demand for that product, the market price drops. Conversely, if there is a substantial demand for the product in the market and a shortage in supply, the market price rises.

The plot in Figure 3.1 is known as a *supply and demand curve*, in which the supply curve is plotted together with the demand curve for a market product. The product price is along the vertical axis and its quantity along the horizontal axis. The solid curve shows the quantity of a product a buyer is willing to buy at a certain price. Similarly, the dashed curve shows the quantity of a product a seller is willing to sell at a certain price. As can be observed in the figure, a buyer is willing to buy more of a product at a lower price and less of a product at a higher price. Naturally, a seller is willing to offer more of his product at a higher price and less of it to a lower price. The point of intersection between the two curves is the ultimate compromise between a buyer and a seller, and is called the *competitive market equilibrium*, accompanied by its corresponding *competitive equilibrium price* P^*, and a *competitive equilibrium quantity* Q^*.

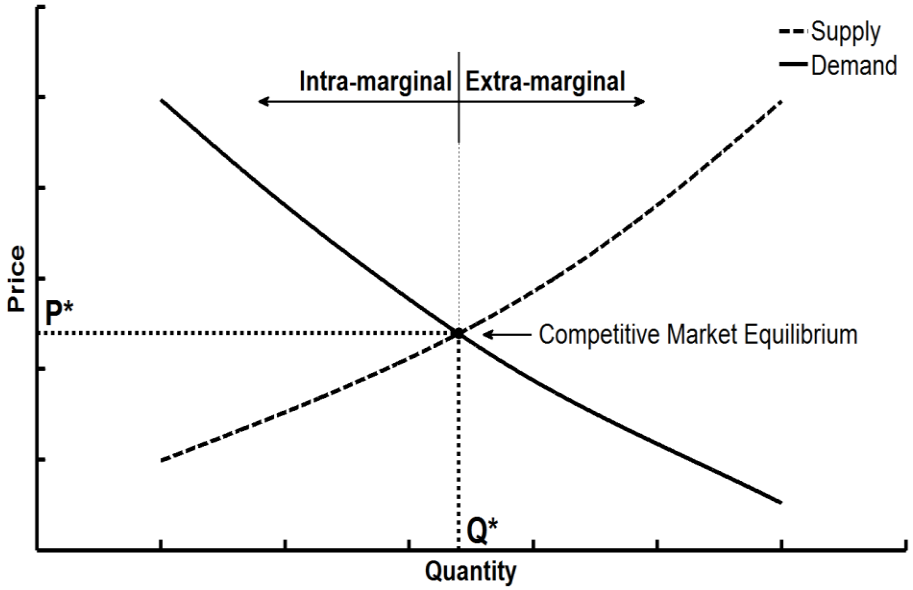

Figure 3.1 The supply and demand curve.

The area enclosed above the market equilibrium and the two curves defines a *surplus* of the product in the market, since sellers are willing to sell more and buyers are willing to buy less of the product. The situation is reversed below the market equilibrium, where a *shortage* of the product results from sellers refraining from selling too much of their product too cheaply and buyers happy to buy more of the product to a lower price.

Buyers and sellers to the left of the market equilibrium are called *intra-marginal traders*, where sellers are willing to sell their products at a cheaper price and buyers are willing to buy products at a more expensive price than the competitive equilibrium price P^*. In this market state, trading occurs frequently between buyers and sellers where the transaction price is P^*. *Extra-marginal traders* are found to the right of the market equilibrium, where the situation is reversed, i.e. sellers' offerings are too expensive and buyers' offerings are too cheap. As a consequence, very little trading occurs in this market state.

The Yin and Yang relationship between supply and demand, sellers and buyers, causes a natural balance between the two forces in financial markets. Therefore, it is only natural that, in a free market, transaction prices converge towards the competitive equilibrium price P^*, at which point the profit of all market participants is maximized, and Q^* is the optimal quantity traded. This micro-economic statement was verified by Vernon Smith in his experiments with human traders in the CDA [7]. He also showed that transaction prices quickly converge to a new market equilibrium following sudden changes in supply and demand (market shock). Vernon Smith's pioneering work in experimental economics earned him the Nobel Prize in Economics 2002.

In 1970, financial markets were claimed to be efficient to such a degree that no excess returns could be made by analyzing historical market data.

3.3 The Efficient Market Hypothesis

In 1970 Eugene Fama released a paper entitled *"Efficient Capital Markets: A Review of Theory and Empirical Work"* [5], based on this PhD thesis [8], in which he proposed three forms of market efficiency; weak-, semi-strong- and strong form market efficiency. Collectively, the three forms are known as the *efficient market hypothesis*, and differ only in what information is factored into prices. The stronger the market efficiency, the more informationally efficient the markets are, and therefore the less chance an investor has in consistently outperforming the market.

Weak form market efficiency states that market prices reflect all past publicly available information and do not exhibit any serial dependencies. This claim necessarily implies that investment strategies based on analyzing and extracting patterns from historical market data is futile. Hence, in the long run, excess returns cannot be made by projecting the past into the future. Instead, market movements rely entirely on current fundamental economic factors.

In semi-strong market efficiency all public information is, once again, included in market prices with the addition that prices instantly, and very rapidly, change to reflect new public information, hence prohibiting any excess returns from being made by trading on

that information. Although, privileged investors can still make a profit from private information, i.e. insider trading.

Strong form market efficiency includes past public and private information in market prices, where prices instantly adjust to reflect new public and private information. This implies that no one can earn excessive returns in the long run, even from insider trading.

For obvious reasons, the efficient market hypothesis created a heated debate in the trading community, where empirical observations and practical experience was used to refute Fama's theory. In practice, markets are not as efficient as Fama's hypothesis implies. This fact is apparent from the multimillion dollar annual returns reported by large financial institutions. Instead, other theories and analysis methods have been successfully employed, as described in the next chapter.

4 Financial Market Analysis

There are two main categories of market analysis an investor can employ in order to determine the state of the current market and to predict its future movement; *fundamental analysis* and *technical analysis*.

In fundamental analysis the state of the global economy, market sectors and individual companies' financial statements are analyzed in order to, for example, value the stock of a company. If the analysis shows that the company's stock price is undervalued by the market, i.e. the stock price as quoted on the exchange is lower than its calculated value from the fundamental analysis, an investor would buy the stock believing that the market will rise. Conversely, if the stock is overvalued by the market, an investor would sell the stock believing that the market will drop in the nearby future.

As the name implies, technical analysis uses a technical approach instead. A technical analyst uses technical charts and technical indicators in his analysis instead of relying on fundamental factors. Charting software usually plots the price of an instrument along a time line. The temporal resolution can usually be configured to show the intra-day (minute, hourly), daily, weekly, monthly or annual price development. The traded volume is often plotted below the price chart. The charting software can also be configured to plot various overlays on top of the price action and technical indicators below the price chart. *Bollinger bands* is a common overlay whereas the MACD (moving average convergence divergence) and RSI (relative strength index) are common indicators. MACD is an example of a trend following indicator, whereas RSI is an example of an oscillating indicator. Indicators are used to determine market trends, overbought/oversold levels amongst other things. Appendix A describes a number of technical overlays and indicators together with their mathematical formulas and sample plots.

4.1 Fundamental Analysis

Fundamental analysis adopts its philosophy from weak-form market efficiency, in respect to not relying on historical market prices to predict future market movements. As its name implies, fundamental analysis is based on fundamental economic theory and studies the underlying economic and political forces that affect the value of an instrument using a

three step process; economic analysis, industry analysis and company analysis [9]. One good example is the calculation of an instrument's *fair value*. For a futures contract on a market index (these terms are described in chapter 4), the fair value expresses the relationship between the futures contract and the underlying market index. If there is a discrepancy between the two, fundamentalist believe that the instrument is either overvalued or undervalued and that the market price will eventually converge towards the fair value. Fundamental analysis can be exercised using either a top-down or bottom-up approach. The three analysis steps, in order, for a top-down approach are; economic analysis, industry analysis and company analysis. For a bottom-up approach, the order is reversed.

The purpose of the economic analysis is to determine the state of the global economy as either expanding or contracting. To this end, economic indicators, such as gross domestic product (GDP), interest rates, inflation and exchange rates are evaluated.

The industry analysis uses the results from the economic analysis to focus on a suitable industry. For example, an expanding economy would minimize the risk of investing in equities such as technology (high risk) whereas utilities (low risk) would be a better choice for a contracting economy. The purpose of the industry analysis is to identify the market leaders and innovators within an industry by assessing market size, total sales, price levels, growth rate, foreign competition and competing products.

The final step is to analyze the health of individual companies by examining business plans, management and financial statements (dividends paid, cash flow, equity, book value, earnings, sales and capital financing). The deliverable from the company analysis is an estimated value of a company's stock. If the estimated stock value is higher than its market value, the stock is currently undervalued by the market (if the analyst's assessment is correct). In such a case, the analyst expects the market price to rise in the near future and eventually converge on the stock's estimated value. For an overvalued asset, the relationship is reversed. In practice, economic ratios are used to assess undervalued and overvalued assets. For stocks, a common ratio is the stock's market price divided by either the company's book value or earnings.

Fundamental analysis is the most effective form of analysis for spotting valuable instruments and predicting long-term trends. Although, fundamental analysis requires good economic skills, market experience and is very time consuming. Therefore, an individual analyst is usually required to focus on a specific industry or group of companies. Furthermore, any assessment made by an individual analyst is highly subjective.

4.2 Technical Analysis

Technical analysis agrees with strong-form market efficiency with respect to all information, public and private, being incorporated in market prices. In addition, technical analysis is based on the belief that market prices exhibit serial dependencies, and hence future market movements can be determined by examining historical market movements. Technical analysts also include philosophies from behavioral finance which

claim that imperfections arise in financial markets due to cognitive- and informational biases. Cognitive biases include factors such as overconfidence and overreaction whereas informational biases relate to predictable human errors in reasoning and information processing. All these factors, including fundamental factors, are believed to be reflected in market prices instantaneously. Therefore, technical analysts claim that historical market prices contain all the necessary information to predict future market movements.

Technical analysis dates back to the 19th century when Charles Dow developed his *Dow Theory*. Dow created the first market indices in the form of two stock market averages; the *Dow Jones Industrial Average (DJIA)* and the *Dow Jones Rails Average*, which is now the *Dow Jones Transportation Average (DJTA)*. He used his two market averages to analyze the behavior of markets in a technical sense. In 1922, Dow's theory was refined and published by William Hamilton in *"The Stock Market Barometer"* [10] and further enhanced in 1932 by Robert Rhea which resulted in *"The Dow Theory"* [11].

Dow theory is based on six basic tenets [12]:

1. *Averages (prices) discount everything*.
   Market prices reflect all available information (public and private) including fundamental economic factors and psychological factors such as the sum total of all hopes, fears and expectations of all market participants.

2. *The market has three trends*.
   Three market trends were identified by Dow; the primary trend (lasting from months to years), the secondary trend (weeks to months) and the minor trend (less than three weeks). An uptrend has a pattern of rising peaks and troughs where each successive rally closes higher than the previous rally high, and each successive rally low closes higher than the previous rally low. The opposite applies to a downtrend.

3. *Major trends have three phases*.
   Trends contain three phases; accumulation, public participation and distribution. A downtrend is associated with "bad news", which is eventually assimilated by the market participants, resulting in an imminent turn in the trend. This is when the most informed investors start taking long positions in the market (buying). This is called the accumulation phase. Most trend-followers begin participating during the public participation phase. This occurs when prices begin advancing rapidly with improved business news. At the peak of an uptrend, public participation and speculative volume increases as more positive economic news is released, resulting in an imminent turn in the trend. This is when the most informed investors start taking short positions in the market (selling) and is called the distribution phase.

4. *The averages must confirm each other*.
   Dow required both his averages to give the same bearish or bullish signals before he accepted a market trend as authentic.

5. *Volume must confirm the trend*.
   To confirm the trend, Dow also required the trading volume to increase in the direction of the major trend. Volume should increase as prices rise and decrease as prices fall in an uptrend. The opposite applies to a downtrend.

6. *A trend is assumed to be in effect until it gives definite signals that is has reversed*.
   Dow argued that a trend will maintain its momentum until an unambiguous signal verifies that a trend reversal is imminent.

Modern day technical analysis is not concerned with *why* prices are at their current levels, only *what* the current price levels are. Usually, prices and trade volume are aggregated over a specific time period to obtain a preferable time resolution for the analysis, such as intra-day (minute, hourly), daily, weekly, monthly and yearly aggregates. Each data point in an aggregate, known as a *bar*, commonly includes the opening-, high-, low- and closing price for the period, together with the aggregated volume (abbreviated as *OHLCV*). The bars are then plotted on technical charts together with overlays and technical indicators to aid in the process of discovering price trends and to time market entry and exit.

As with fundamental analysis a top-down or bottom-up approach can be used in technical analysis. A top-down approach starts with an identification of the overall trend. Trends can be spotted in technical charts by constructing trend lines, drawing moving averages and performing peak/trough analysis. Support lines are used to mark areas of congestion of previous lows below the current price mark support levels. A break below the support line would be considered bearish. Similarly, resistance lines are used to mark areas of congestion of previous highs above the current price mark resistance levels. A break above the resistance line would be considered bullish.

Next, the momentum of the trend, buying/selling pressure and the relative strength of a security is measured by using technical indicators. Trend momentum can be measured with, for example, the MACD indicator. MACD is based on closing prices and is constructed by subtracting the 26-period Exponential Moving Average (EMA) from the 12-period EMA. Furthermore, the 9-period EMA of the MACD indicator, known as the MACD signal line, is calculated and plotted in the same chart as the MACD indicator. Signal line crossovers can be used to identify turns and points of market entry/exit. As an additional step, the MACD-histogram can be created by subtracting the signal line from the MACD indicator. A positive histogram (above the zero line) represents a bullish momentum and a negative histogram a bearish momentum. Buying/selling pressure can be measured with, for example, the Chaikin Money Flow (CMF) oscillator, which incorporates trading volume together with closing price calculations. CMF fluctuates

between -1 and +1. When CMF is above zero, buying pressure is dominant (bullish), and below zero selling pressure is dominant (bearish). The relative strength of a security is commonly measured by dividing the security by a benchmark. For stocks, the price relative is usually formed by dividing a stock with its stock index (e.g. the S&P 500). Plotting the price relative will reveal if the stock is outperforming (rising) or underperforming (falling) its index. Conceptual and mathematical descriptions of some common technical indicators are provided in Appendix A.

The technical analysis results in an estimation of the strength and maturity of the current trend (if one exists), from which the reward-to-risk ratio and entry levels for new positions can be determined.

4.3 Random Walk Theory

A treatment of financial market analysis would not be complete without mentioning *random walk theory*. Random walk theory supports the semi-strong efficient market hypothesis, thereby opposing the notion of consistently outperforming the market by basing trading decisions on past market prices. In 1973, Burton Malkiel released the first edition of his book *"A Random Walk Down Wall Street"* [13], based on random walk theory, developed by British statistician Maurice Kendall and published in his paper *"The Analysis of Economic Time Series – Part I: Prices"* [14]. Malkiel theorized that prices have no memory, are totally random and fluctuate randomly about their intrinsic value. Therefore, they cannot be used to predict future prices. According to Malkiel, news events are also totally random. As a consequence of random walk theory, the only rational approach of trading financial markets is by employing a buy and hold trading strategy and crossing one's fingers.

4.4 Computational Intelligence in Financial Engineering

Both fundamental and technical indicators can be incorporated as feature vectors using a machine learning approach. Just as a human trader can base his trading decisions on trend following and oscillating technical indicators, a predictive model can be trained to generate trade signals based on patterns discovered in indicators. This is the approach employed in this paper.

Other approaches use agent theory, from the field of artificial intelligence, where a software-based agent continuously receives signals (percepts) from its environment in the form of market price and volume updates, and reacts by outputting signals to buy or sell a certain quantity of an instrument. A software agent can of course calculate technical indicators internally, but the core logic is usually based on learning, inferring new rules, planning and making decisions based on its sensory inputs (percepts) from its stochastic environment.

As was mentioned in the introductory chapter, stochastic calculus can also be used to create quantitative models of stochastic processes. Independent of any approach used, a basic knowledge of financial markets and the process of trading them is needed in order implement computational intelligence in practice. This chapter, including the previous chapter, provided basic concepts of financial markets and methods for analyzing them.

The next chapter provides some basic terminology and concepts involved in trading financial markets.

5 Trading Financial Markets

The aim of this chapter is to introduce some basic concepts relating to trading financial markets and is not intended as a rigorous treatment of financial calculations, risk management, portfolio theory or the history of trading. The purpose of the chapter is to further add to the knowledge base in order to understand the problem that will be presented to the machine learning technologies in the following chapters and to familiarize the reader with some relevant terminology.

5.1 Assets

Trading is all about making investments, thereby taking on risk, in the hope of obtaining a profitable return. In order to make an investment an *asset* needs to be traded. An asset can be defined as a resource with economic value that some entity owns in the hope that it will provide a future benefit. Assets can be divided into three main classes; equities (stocks), fixed-income (bonds) and cash equivalents (money market instruments). In practice, commodities (physical products and materials such as gold, oil, wheat and cattle) and real estate are also added to the definition.

5.2 Indices

Often major financial entities create an *index* for a specific asset class. For example, the S&P (Standard and Poor) 500 is a stock index based on 500 US companies. The "Standards and Poor" means that the index is made up of both large and small companies. An index can be calculated in different ways, but its main purpose is to define a benchmark of its constituents to measure their performance (essentially, the S&P 500 is a well diversified portfolio). In this respect, the S&P 500 is a leading indicator for US equity (stocks) and measures the overall performance of US companies.

5.3 Derivatives

A *derivative* can be defined as a financial instrument whose value depends on more basic underlying variables. Often the underlying variable for a derivative is the price of an asset. For example, a stock option is a derivative of a stock. Its price is based on (derived from) the price of its underlying stock asset. Derivatives come in multiple flavors, the most common being *futures* and *options*. A derivative can even have an index as its underlying instrument, such as the S&P 500 index futures contract.

In order to understand futures and options, let's first define a *spot market*. A spot market reflects the current price of an asset. If, for example, you wanted to buy some stocks in Apple Inc, you could buy them, right now, for the current price in the spot market.

In a *futures market*, on the other hand, you buy and sell contracts for an instrument at a certain price in the future. One common example used to understand the concept of futures markets, is that of a farmer who expects to harvest his crop at some future date. The farmer, who is growing wheat, knows that he will have to harvest his crop three

months from now and sell it at the current price level at that future date. Although, the farmer believes the market price for wheat will decrease during the three month period, so he sells a certain amount of future contracts of wheat in the futures market three months ahead of time. On the other side of the deal, there is of course a counterparty who believes the price of wheat will rise, so the counterparty buys the future contacts instead. If the farmer's hunch is correct, the price of wheat decreases over the next three months, and when the future contracts expire at the end of the third month, he can now sell his wheat to the counterparty for the price agreed upon three months ago. This price will be higher than the current price level and therefore the farmer pockets a profit (or has effectively avoided a loss).

An *option* is similar to a future, but the holder of an option has the right (option) to back out of his deal. For example, if the farmer had used options instead of futures, and the wheat market had risen during the three months, he could have opted not sell his wheat contracts. The downside to an option is that a small fee has to be paid when the contract is obtained. Conversely, a futures contract does not incur a fee when the contract is obtained, but the holder does not have the right to opt out of his deal when the contract matures.

5.4 Contracts

Some new terms were introduced in the text above. A *contract* defines what underlying asset is being traded, at what price, the quantity of the underlying asset and the *delivery date* (or *maturity date*). The delivery or maturity date is also called the *expiration date* for a derivative and is the date at which the contract is realized, i.e. when one party sells the underlying asset and the other party buys it. For commodities, two additional pieces of information are included in the contract; the *quality* (*grade*) of the underlying asset and the *delivery procedure*. The underlying asset for a commodity derivative is something physical, for example oil, gold, gas and wheat. Therefore, commodity derivatives usually involve a physical delivery process when the contract expires, i.e. the seller must deliver a certain amount of the underlying asset to the buyer at a certain time and place. For derivatives on financial instruments, such as stocks, bonds and foreign exchange, the underlying asset is usually just a number in a computer, so no physical delivery is required. Such contracts are usually *cash settled*, i.e. one party hands over cash to the other party in the form of an electronic transfer.

Derivative contracts can have different delivery schedules, which is also defined in the contract. The most common schedules are; monthly, quarterly, semi-annually and annually. As an example, the S&P 500 index futures contract has a quarterly schedule with deliveries in March, June, September and December every year.

5.5 Over-The-Counter (OTC) Markets

When large financial institutions trade with each other, and with their clients, they preferably do so in the Over-The-Counter (OTC) market. In an OTC market, the contract is defined ad hoc, i.e. both parties agree upon the specific details of the contract, such as the underlying asset, the quantity of the asset, the price, delivery, etc. When the contract expires, the deal is settled, and both parties exchange goods and money with each other.

In an OTC market, there is always some *credit risk* involved. Credit risk essentially means that one party might not honor his end of the deal when the contract expires. This does not usually happen, since it would damage the business relationship between the two parties for future deals, but the credit risk is always present. The type of derivatives traded in an OTC market are usually *forwards*, *swaps* and *swaptions* (*swap options*). A forward is very similar to a future contact, but is only traded between large financial institutions. A swap is exactly what it sounds like, i.e. two parties agree to swap, for example, two currencies at the beginning of the contract period (e.g. GBP for USD), and when the contract matures, the two currencies are swapped back again. A swaption is essentially a option on a swap contract.

5.6 Exchange-Traded Markets

In order to stimulate trading for investors, exchange-traded markets were created. An exchange-traded market is a market that is traded through an exchange, such as the CBOT (Chicago Board of Trade), CBOE (Chicago Board Options Exchange), CME (Chicago Mercantile Exchange), NYMEX (New York Mercantile Exchange) and the NYSE (New York Stock Exchange). Exchange-traded markets are *regulated*, provide *standardized contracts* and remove *credit risk*.

Markets in the US are regulated by bodies such as the CFTC (Commodity Futures Trading Commission) and SEC (Securities and Exchange Commission) to prevent fraud and other actions that might have a destabilizing effect on the economy, e.g. *cornering the market* and *front running*.

Standardizing contracts makes trading easier for both parties. Exchange-traded markets offer standardized contracts, in which all the contract details are unambiguously defined such as the underlying asset, the minimum amount the price of a contract can be changed (tick size), the amount of the asset included in one lot of the contract (contract size), the unit of measurement (e.g. barrels of oil, ounces of gold or bushels of wheat), contract months (e.g. crude oil futures contracts might trade on a quarterly basis, i.e. with expiries in March, June, September and December), delivery details, etc. Using standardized contracts, each party knows exactly what it means to buy or sell 1 contract of e.g. crude oil.

Credit risk is minimized in exchange-traded markets by using something called margin accounts.

5.7 Margins, Leverage and Fees

A *margin account* is an account that is maintained by an exchange for one of its clients (exchange member) in order to ascertain that a client will honor his deal. In other words, margin accounts effectively reduce credit risk. If a client is not an exchange member, margin accounts are provided through a broker (the broker, in turn, is an exchange member and therefore owns the physical margin account). When an investor wants to start trading derivatives on an exchange-traded market, he opens a margin account by depositing an initial sum of money, the *initial margin*. Every time the investor enters an order into the market to buy or sell one contract, he needs to have a certain amount of

money in his margin account per contract. This mechanism ensures that an investor can honor a deal.

The amount of money per contract in the investor's margin account is not the full price of the contract. This is what is meant by *leverage* in a derivative market, since an investor can buy or sell derivative contracts to a fraction of their full value. The leverage that is provided in derivatives markets necessarily means that an investor can accumulate larger profits, or incur larger losses, than in a spot market.

Futures contracts are *settled daily* and not at maturity. This means that if the market moves in favor of the investor during the trading day, the profit will be deposited in his margin account before trading commences the following trading day. Conversely, if the market moves against the investor during the trading day, the loss is deducted from his margin account. Other fees, such as broker *commission fees*, *transaction fees* and *exchange fees*, are also deducted from the margin account. If the margin account drops below a certain threshold, known as the *maintenance margin* (which is lower than the initial margin), a *margin call* is issued, whereby the investor is expected to top up his margin account to the initial margin during the following trading day. If this is not respected, a broker has the right to close out enough contracts to the current market price in order to rebalance the investor's margin account.

5.8 Electronic Exchanges

Twenty years ago, trades were negotiated by *pit-traders* in something called the open-*outcry*. Traders in colorful jackets would gather in an area in the middle of the trading floor that looked like an open pit. When a client called in an order, it was received by a broker, who scribbled down the order details on a piece of paper and handed it over to a *runner*. The runner would literally run between the broker and the trading pit, where he would hand over the paper slip to a trader. Using a special kind of sign language and a loud voice, the trader would then negotiate a trade with another trader. Appropriately, this type of trading environment was called the pit-traded open-outcry.

The most common trading environment today consists of electronic exchanges, where the investors, acting as traders, enter their own orders into the market using a computer. The order is routed to the electronic exchange where it is queued in either a bid (buy) queue or an ask (sell) queue. An automated order-matching algorithm then pairs bid and ask orders appropriately in order to produce trades between two parties. A trader receives trade confirmations, price updates and other market information via the electronic exchange's API (Application Programming Interface) and usually has some visualization software installed on his computer that plots price curves and technical indicators on the trading screen. Financial news feeds are also provided electronically via one of the major news agencies (e.g. Reuters, Bloomberg).

5.9 Trading Styles

There are three main categories of traders in financial markets; *hedgers*, *speculators* and *arbitrageurs*.

Hedgers do not trade the markets with the intension of making a profit. Their main intension is to reduce risk. By *hedging a position*, a trader places an opposing order into the market to offset his risk of a future market movement. For example, in June the treasurer for a US company knows that the company will be paying £10 million in September for goods purchased from a British supplier. The USD/GBP exchange rate offered by a financial institution is 1.6281 (bid) / 1.6285 (offer) in the spot market and 1.6187 (bid) / 1.6192 (offer) for the 3-month future contract. In order to lock in the exchange rate and protect the company from adverse market movements during the three month period, the treasurer therefore buys sterling (GBP) from the financial institution in the 3-month future market. This fixes the price to be paid to the British company at $16,192,000. Hedging minimizes loss if the market moves against the investor, but also minimizes any profit if the market moves in favor of the investor.

Speculators are the equivalent of professional gamblers. They are willing to bet on a future market movement by placing orders accordingly. For example, using the exchange rates from the previous example, a speculator who believes the financial institution has undervalued the exchange rate for the 3-month contract, i.e. that believes the exchange rate will be higher in September, can buy a certain amount of the 3-month futures contract in June and sell it to a profit in September if the market does in fact rise during the 3-month period. Although, the speculator runs the risk of incurring a loss if the market moves in the opposite direction.

A further categorization of speculators is; *scalpers*, *day traders* and *position traders*. Scalpers trade the volatility (small, short-term price variations) and short-term trends in the market and normally only hold positions in the market for a few minutes before closing them out. Day traders hold their positions for less than one trading day, i.e. *intra-day positions*, in order to avoid any market movements between trading days (the opening market price during one trading day can be different from the previous trading day's closing price if, for example, bad financial news is released overnight). Position traders hold positions in the market for much longer periods, in the hope of making profits from major market movements.

Arbitrageurs are in the business of locking in riskless profit by taking opposing positions in two different markets. They continuously monitor markets for any discrepancies in price levels. For example, when the IPE (International Petroleum Exchange) in London transitioned from the open outcry to an electronic exchange, it offered crude oil contracts in both markets. During a short period, it turned out that crude oil was offered to a cheaper price in one of the markets. This arbitrage opportunity meant that a trader could buy the cheaper crude oil contracts in one market and sell them to a higher price in the other. This particular type of arbitrage is called *pure arbitrage*. The same discrepancy materialized between the IPE and NYMEX for crude oil when NYMEX started an electronic exchange a couple of years later. Arbitrage opportunities are only present for short periods of time, especially nowadays when statistical arbitrage trading, performed by high-frequency trading systems, is common. In high-frequency trading, high-performance, automated trading systems continuously scan multiple markets for arbitrage opportunities and place thousands of orders per second. Due to the law of supply and

demand, high-frequency systems effectively wipe out any arbitrage opportunities between markets within seconds.

5.10 Order Types

The three most common types of orders traded on electronic exchanges are; *market orders*, *limit orders* and *stop-loss orders*. In order to understand order types, we need to first understand the *order book*, *market spread* and *market depth*.

The *order book* is displayed as part of a trading screen and is structured as a table, in which the columns are commonly laid out in the following order (from left to right): *bid quantity*, *bid price*, *ask price* and *ask quantity*. There are of course more columns displayed on a professional trading screen, but these four columns will suffice in order to understand the essentials. The middle row in the order book displays the best bid and best ask with corresponding bid and ask quantities. For example, a market might have 10 buy (bid) orders offered at $100.50 and 20 sell (ask) orders offered at $102.50 as the best bid and ask as shown in Figure 5.1 (middle row). The quantities show the combined orders that are offered at a specific price level. The price difference between the best bid and best ask is called the *market spread*.

ORDER BOOK				
CONTRACT	BID QTY	BID PRICE	ASK PRICE	ASK QTY
ESU1			104.00	30
	10	100.50	102.50	20
	15	99.50		

Figure 5.1 Order Book.

Now, picture other investors placing bid and ask orders, offering to buy and sell at price levels slightly cheaper (bid orders) and expensive (ask orders) than the best bid and ask. If the more expensive ask orders are piled on top of each other above the best ask in increasing offer price order, and the cheaper bid orders are piled underneath each other under the best bid in decreasing price order, the resulting order book shows the *market depth*. In other words, the best bid and ask is shown in the middle of the order book and the market depth is shown below the best bid and above the best ask respectively. Let's say the next best bid price is $99.50 with a corresponding quantity of 15 and the next best ask price is $104.00 with a quantity of 30. The order book would then have one middle row containing the best bid and ask price and corresponding quantities with an additional row beneath the best bid and one additional row above the best ask. The market depth displayed is therefore 2, with two levels of bids and asks respectively (see Figure 5.1).

Switching back to the order types, a *market order* is a request to carry out a trade immediately at the best available price in the market. Using the order book described

above, if a market order to sell 15 contracts is floated into the market, it would immediately trade with 10 contracts at the best bid in the market. Since 5 contracts of the market order are still outstanding, 5 contracts will also be traded at the next best bid price (market depth 2). Furthermore, since all the contracts in the order book for the best bid were traded, the next best bid now becomes the best bid with a price of $99.50 and a remaining quantity of 10.

A *limit order* also specifies a quantity, but with the addition of a price. A limit order will only be executed at the desired price level or at a more favorable price level for the investor. If a limit order to sell 15 contracts at $102.00 was placed into the market, it would enter the sample order book as the best ask and remain idle until the market rises to $102.00, at which point it will cross (trade) with the best bid. Similarly, if the price for the limit order had been $105.00, the limit order would enter the order book as the top entry in the ask stack at a price level $104.00 and therefore be displayed at market depth 3.

A *stop-loss order* is used as an insurance for adverse market movements. Imagine a trading scenario where you as the speculator believe the market will rise during the trading day. Therefore, as your initial action, you place a limit order to buy 10 contracts of, for example wheat, for $101.50 each. Eventually the market starts rising and your limit order crosses with a counterparty's sell order for 10 contracts. In the parlance of trading, you are now said to be *long* 10 contracts of wheat, whereas your counterparty is said to be *short* 10 contracts of wheat. It follows that a trader holding a long position will benefit from a rising market, since he can sell his contracts at any point at a higher price than he bought them for. This specific trading strategy is called a *buy-and-hold* strategy, since you buy a certain amount of contracts, hold on to them in the hope the market will rise, and sell them at a higher price when the market has risen. Let's say the market rises to $105.00. You can now sell your 10 contracts and make a profit of ($105.00 - $101.50) * 10 = $35.00. But what if the market suddenly drops to $98.00 before you can react? If you sell your contracts now you would incur a loss of ($98.00 - $101.50) * 10 = -$35.00. This is where a stop-loss order comes in handy. When you place your initial limit order to buy 10 contracts at $101.50, you can place a stop-loss order to *sell* 10 contracts at, for example, $100.00. Following the scenario above, when the market suddenly drops to $98.00 your stop-loss order would kick in at $100.00 and cut your losses short, i.e. instead of losing $35 (or even worse if the market drops even more) you have only incurred a loss of ($100.00 - $101.50) * 10 = -$15.00.

This chapter concludes the theory of financial concepts, financial market analysis and trading financial markets. The next chapter introduces the general process involved in mining information from data and creating predictive models, followed by three chapters describing specific machine learning technologies that can be used to create predictive models of financial markets.

6 The Data Mining Process

Since the birth of the scientific method, following the 17th-century scientific revolution, scientists have employed experimentation and empiricism in order to evaluate hypotheses

and to gain knowledge from data. Back in the early days, data was scarce and therefore statistical methods were developed in order to obtain as accurate results as possible from limited data samples. Nowadays the situation is reversed due to advances in data acquisition and storage technology, which has enabled an exponential growth in the availability of data. Therefore, new methods and techniques for processing and extracting useful information from huge amounts of data have been developed and are organized under the common term *Data Mining*. Data mining is a confluence of various techniques, including statistics, artificial intelligence, machine learning and database technology. The multidisciplinary nature of data mining is illustrated in Figure 6.1.

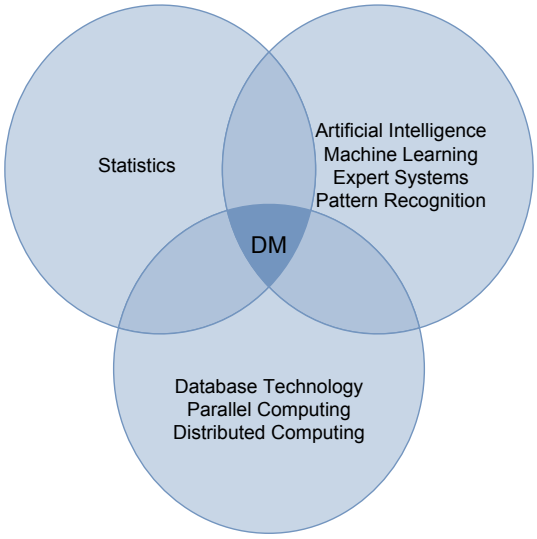

Figure 6.1 The multidisciplinary nature of Data Mining.

During the last decade, various processes have been developed for performing data mining, some being more elaborate than others. The simplest form consists of the three steps; preprocessing, data mining and postprocessing. A more detailed process is the *knowledge discovery in databases (KDD)* process [15] which is visualized in Figure 6.2. The figure shows the incremental and iterative nature of the process.

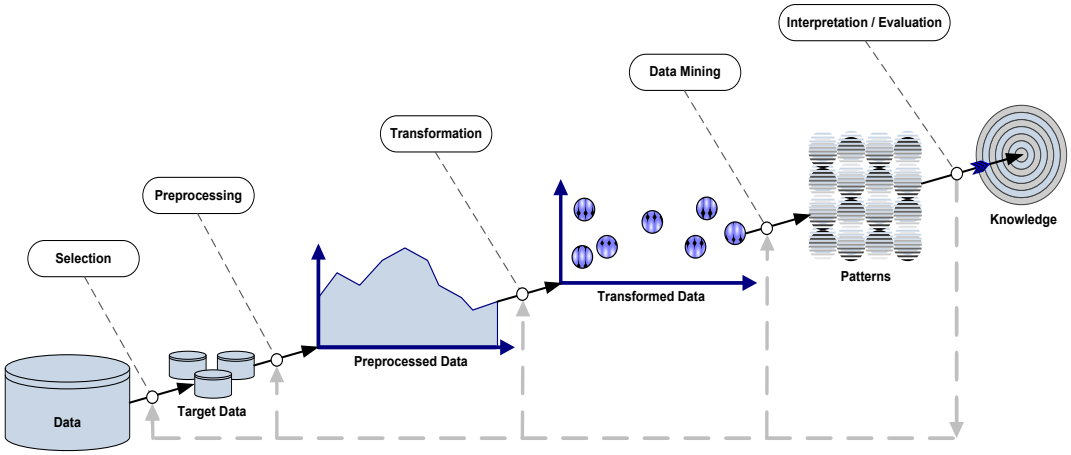

Figure 6.2 The Knowledge Discovery in Databases (KDD) process.

The standard data mining process used today is the *cross industry standard for data mining (CRISP-DM)* process [16] and is depicted in Figure 6.3. The arrows within the outer circle show transitions between the various phases in the process, with *Business Understanding* as the initial step. The outer arrows along the circumference of the circle symbolize the ongoing nature of the process, that is, once models have been deployed and monitored, information is fed back into the process.

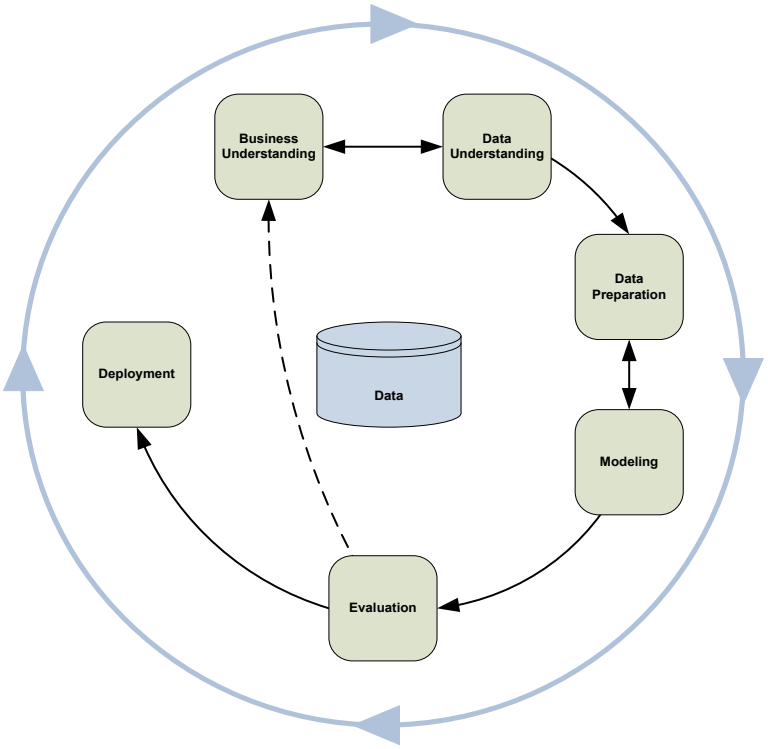

Figure 6.3 The Cross Industry Standard for Data Mining (CRISP-DM) process.

There is a natural mapping between the three data mining processes as listed in Table 6.1. As the table shows, CRISP-DM is the most complete.

Table 6.1 Data Mining Process Mapping

Simple	KDD	CRISP-DM
		Business Understanding
Preprocessing	Selection	Data Understanding
	Preprocessing	
	Transformation	Data Preparation
Data Mining	Data Mining	Modeling
Postprocessing	Interpretation / Evaluation	Evaluation
		Deployment

The phases in the CRISP-DM process are summarized in Table 6.2 and subsequent chapters describe each phase in more detail, with a focus on process tasks and data mining concepts relevant to this paper, that is, predictive classification techniques within a financial context.

Table 6.2 CRISP-DM Phases

Phase	Description
Business Understanding	The first phase concentrates on understanding the problem domain, its requirements and its objectives from a business perspective. This knowledge then needs to be formulated as a data mining problem within the constraints of the requirements in order to achieve the objectives.
Data Understanding	The second phase starts with identifying and collecting initial raw data. The most important tasks in this phase include getting acquainted with the data, understanding the data, identifying quality issues with the data and deriving hypotheses from the data. Data understanding is highly explorative in nature.
Data Preparation	The intention with the data preparation phase is to prepare a complete dataset for the modeling phase. The phase involves activities such as data cleaning, attribute selection, data transformation, data normalization and data dimensionality reduction. Some modeling techniques require a certain format of the dataset which might necessitate a proactive approach during this phase.
Modeling	This phase involves selecting appropriate modeling techniques and applying them to the dataset. During the modeling phase, models are created and their parameters are optimized by validating the quality of the models.
Evaluation	Before the models are deployed, they need to be tested and their performance and generalization behavior needs to be evaluated. Furthermore, requirements and objectives are traced back to the initial phase to ensure that the deliverables are in line with the business goals.
Deployment	The final phase in a complete iteration consists of deploying the deliverables. This might just equate to producing a document containing newly obtained knowledge from the data mining project. Or, it could involve deploying a model into a live environment. Another important task during this phase is to review and document all steps carried out during the iteration for future projects.

6.1 Business Understanding

In a commercial knowledge discovery and data mining project, the business requirements, business objectives and success criteria need to be fully understood. The business problem also needs to be formulated as a data mining problem and a project plan needs to be produced. Resources planning, resource procurement and allocation, budgets, risk assessments and contingency plans are also part of a commercial effort.

In this paper, a complete and thorough treatment of the business understanding phase is out of scope. Instead, it is reduced to understanding the problem domain, which is supported by chapters 3, 4 and 5 and further elaborated on in chapter 12.

6.2 Data Understanding

The data understanding phase consists of collecting an initial dataset for explorative purposes. The structure and organization of the data is also described and an assessment is made with regards to its quality.

6.2.1 Collecting Initial Data

Once the business requirements and objectives have been understood and the business problem has been formulated as a data mining problem, appropriate data sources are indentified and an initial data sample is collected. Obviously, the exact data sources depend on the problem domain.

In the context of the problem domain addressed in this paper, appropriate sources include commercial and free providers of historical financial data. All major exchanges provide such data. For example, The Chicago Mercantile Exchange (CME) offers historical data for all derivatives traded on their Globex electronic platform. Other leading, commercial suppliers of financial market data are the major news agencies, such as Reuters and Bloomberg. The two most prevalent free sources are Google Finance and Yahoo Finance.

6.2.2 Describing the Data

Describing the data involves figuring out what is contained within the initial dataset and how it is organized. Different data providers use different data formats, the most common being a tabular format of rows and columns, where each row represents an object and the columns constitute different attributes (properties) for the objects. The most important characteristics of the dataset are the nature of its attributes, the number of attributes and if the data is ordered or not.

6.2.2.1 Attributes

In 1946 Stanley Smith Stevens released his theory of scales of measurement [17]. In his paper, he claimed that all measurement in science could be classified into four different types of scales; *nominal*, *ordinal*, *interval* and *ratio*. The same classification applies to attributes, where nominal and ordinal attributes are grouped under *categorical attributes*, with a similar grouping of interval and ratio attributes into *numeric attributes*.

Categorical attributes are qualitative in nature, where nominal attributes are just names of things, used for mutual exclusive, non-ordered categorization, whereas ordinal attributes are used when order does matter, but not the difference between values. Hence, the mathematical operators that can be applied to nominal attributes are equality and inequality ($=,\neq$), whereas ordinal attributes can be ordered by including the remaining two comparison operators ($<,>$).

Numeric attributes are quantitative in nature and take on real or integer values. Interval attributes provide a unit of measurement where differences between values makes sense. They adopt the four comparison operators and include the mathematical operators of addition and subtraction (+,-). Ratio attributes are used when ratios between different objects are meaningful and include all the mathematical operators described above with the addition of multiplication and division (*,/).

Attributes can also be either *discrete* or *continuous*. The domain of continuous attributes is the entire real number scale, hence, they have an infinite range. Discrete numbers have a finite, or countably infinite, range. When an attribute consists of discrete numbers containing only two different values, it is called a binary attribute.

6.2.2.2 Ordered Data Sets

In some datasets, the order in which the records appear has no consequence for processing purposes. A good example is the iris dataset containing length and width measurements for the petals and sepals of different classes of flowers [18]. Processing the dataset with the intention of finding correlations between the various attributes and the different flower classes can be done in any order.

Other datasets contain ordered or sequential data, in which the processing order does matter. Datasets containing spatial or temporal data, for example the various temperature measurements at different geographical locations over a period of time, is an example of an ordered dataset. Such data contains samples where two measurements, close in space and time, are similar to each other (autocorrelation). In order to determine how the temperature varies with location and time, the data needs to be processed sequentially. Financial time series fall into this category since they have both a spatial component (price and volume) and a temporal component (time). Even if the time series is transformed from the time domain into the frequency domain (using the Fourier transform) to analyze its spectral content, the various frequencies follow the same constraints.

6.2.3 Exploring the Data

The goal of exploring the data is to get a better understanding of the data. For this purpose, descriptive statistics and visualization are commonly used. Using descriptive statistics the data can be summarized by calculating, for example, its central tendency (mean, median, mode) and dispersion (variance, standard deviation, percentiles). In some cases, calculating the distribution of the data and relating it to a known statistical distribution can be of importance for further processing.

Visualization techniques include plotting the complete data, parts of it, or combinations of attributes. Various types of plots are commonly included in data mining software packages, such as, scatter plots, box plots, histograms, contour plots, surface plots and parallel coordinates. Specialized plots are also common for specific application domains. A wealth of information and understanding can be obtained from appropriately constructed visualizations.

6.2.4 Verifying Data Quality

One important concern of the initial data exploration is to assess the quality of the data. Data quality is crucial for the outcome of the data mining project. Incorporated into the definition of data quality issues, is the appropriateness of the data for the data mining problem, the amount of missing and duplicate values, and the amount of noise in the data. The data can be contaminated in other ways and include discontinuities for sequential data. When the data is fused together from multiple sources, two different attributes could

mean the same thing (synonym) or two attributes with the same name could mean two different things (homonym). Data quality is a large area of research and the process of cleaning contaminated data is very time consuming.

6.3 Data Preparation

In the data understanding phase, an initial data sample was collected, described and explored in order to gain a better understanding of the data and to assess its quality. In the data preparation phase, the final, raw dataset is selected, cleaned and transformed into a more suitable format for the modeling phase. Since the initial data sample also needs to be preprocessed in some way, the preprocessing of the initial and final datasets is usually combined in the data understanding phase, or the final dataset is subjected to a similar preprocessing step in the data preparation phase. In any case, the various preprocessing techniques are described in this subchapter.

6.3.1 Selecting the Data

With support of the initial investigation conducted during the data understanding phase, an appropriate, raw dataset is selected. The rationale behind selecting the specific dataset is usually motivated and document as an initial task in the data preparation phase.

6.3.2 Cleaning the Data

As was briefly mentioned in the data understanding phase, the selected dataset needs to be cleaned in order to ensure high data quality for the data mining phase. This involves, amongst other things, dealing with noise, artifacts, duplicate data, outliers and missing values.

Noise and artifacts can be present in the data in various forms and are highly dependent on the application area. In order to present a more intuitive example of identifying and removing noise and artifacts from a dataset, an example from biomedical signal processing of EEG data will be used.

Bioelectric signals have a much smaller magnitude and lower frequency compared to the electrical grid used for powering electrical appliances. When measuring the bioelectric activity in the human brain using an electroencephalogram (EEG), this poses a problem since the more powerful electromagnetic fields emanating from the power grid contaminate the sensitive readings from the EEG scalp electrodes. What happens, is that an electrical potential is transferred into the EEG equipment via electromagnetic induction, which overpowers the measurement of the biosignals. Two common noise components that need to be filtered out are; baseline wander and the 50Hz (or 60Hz depending on country) noise caused by the intrinsic frequency of the power grid. A ten minute sample of an EEG is shown in Figure 6.4. The two top plots show the contaminated signal and the two bottom plots the filtered signal. The left plots show the signal in the time domain (amplitude versus time) and the two right plots show the signal in the frequency domain (frequency versus time). Comparing the two left plots, the baseline wander can be seen in the top plot, where the signal displays an uptrend to the right. This is caused by a contaminating direct current (DC) component. In the lower plot, the baseline wander has been eliminated by using a high pass filter. Comparing the two

right plots, the noise caused by the power line can be observed at 50Hz (actually, this sample is relatively noiseless in this respect and does not have a spike at 50Hz which it normally would). In the lower plot, the 50Hz noise component has been filtered out with a notch filter, which shows up as a small "dip" in the spectral plot at 50Hz.

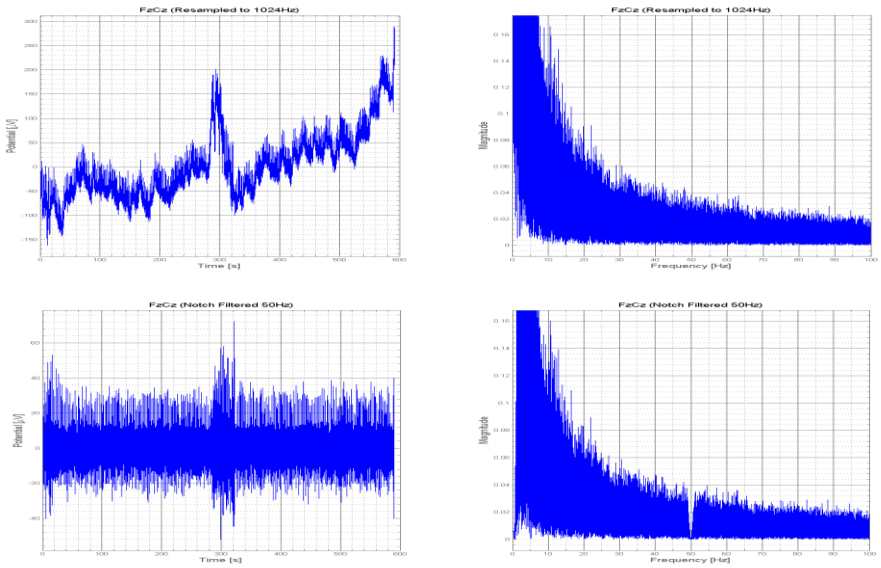

Figure 6.4 Time series with noise (top) and without noise (bottom).
Temporal plot (left), spectral plot (right).

When measuring someone's EEG, it is humanly impossible for that person not to blink their eyes every now and then. When doing so, motor neurons in the nervous system produce bioelectric signals causing the eyes to blink. This contaminates the measurements of other bioelectric activity in the brain. Eye blink artifacts show up as small transient spikes in the EEG recording, but can be effectively removed by singling-out the artifact components using independent component analysis (ICA) and back projecting the artifact-free components on the original signal. Figure 6.5 shows a 10 second amplitude plot of EEG measured by two different scalp electrodes. In the red plot, the eye-blink artifacts are remarkably visible, whereas in the blue plot they have effectively been filtered out.

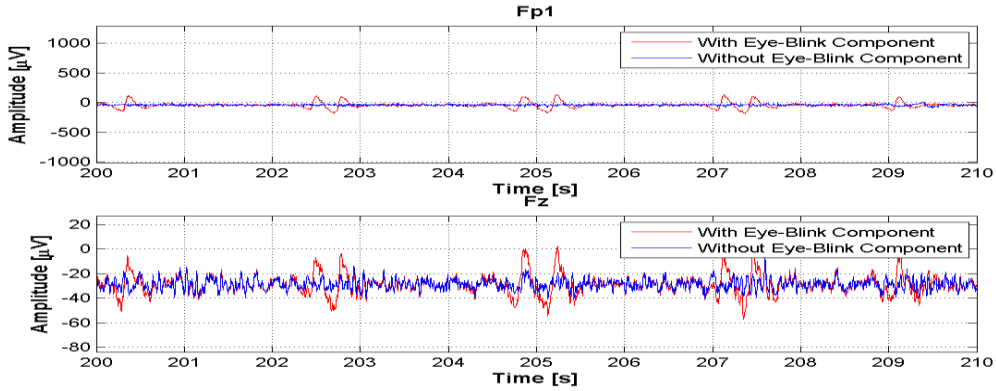

Figure 6.5 Time series with artifacts (red) and with artifacts eliminated (blue).

An enormous amount of data is available from many diverse data sources. This has its pros and cons. The pros include the wealth of information and knowledge that can be harvested by combining the various data sources, i.e. through data fusion. The cons include, amongst other things, the possibility of duplicate data. The issue of duplicate data arises when two or more data sources containing identically named attributes are merged with each other. Another possible conflict arises when two identical or similar rows end up in the same dataset. If the rows differ with respect to one or more attribute values, conflicting information might be present. One example is two rows identifying the same person, but with two different surnames (e.g. maiden name and married name). Duplicates can be handled by deleting conflicting rows or by modifying attributes in conflicting records.

Datasets can also contain missing attribute values for one or more records. Depending on the nature of the data, missing values can be handled by interpolating between existing values on either side of the missing values in sequential data or by replacing categorical values with representative values. Another option is to delete rows containing missing values.

Depending on the application, outliers can also be removed from the dataset or excluded when processing the dataset. An outlier is defined as a data point that differs exceptionally from the average set of data points. In fraud detection applications, outliers are the main focus of attention and cannot be removed, since they usually imply fraudulent activity.

6.3.3 Constructing the Data

The task of constructing the dataset involves choosing appropriate attributes for the modeling phase by selecting individual attributes or deriving new attributes by combining two or more attributes. It also involves extracting application specific features from the dataset and transforming the dataset.

6.3.3.1 The Curse of Dimensionality

One major concern in data mining is the dimensionality of the data, i.e. the number of attributes (features) in the dataset. Some modeling techniques perform poorly if the dimensionality of the data is too large. A dataset containing many dimensions makes the data points sparse in the space that they occupy, which especially constitutes a problem for modeling techniques incorporating distance and density measures in their algorithms, such as k-means, k-nearest neighbor and DBScan. A smaller dimensionality also reduces the time and memory complexity of data mining algorithms. In general, a smaller dimensionality, as opposed to an exceptionally high dimensionality, yields better model performance. Therefore, reducing the dimensionality of the data is an important task during the data preparation phase. The issues caused by high dimensionality in datasets is commonly known as *the curse of dimensionality*.

6.3.3.2 Feature Construction

One simple measure to reduce the dimensionality of the dataset is to derive new attributes by combining two or more attributes. This is called *feature construction*. For example, a dataset containing the two attributes, mass and volume, could be replaced by the single derived attribute, density = mass / volume.

6.3.3.3 Feature Subset Selection

Instead of combining attributes in order to reduce the dimensionality of the data, a subset of the attributes can be selected. This is called *feature subset reduction* or *feature subset selection*. This method can also be used to eliminate redundant and irrelevant features.

Feature subset reduction can be accomplished by using either *embedded approaches*, *filter approaches* or *wrapper approaches*. As the name implies, in embedded approaches feature subset selection is intrinsic to the data mining algorithm. Decision trees are a good example, which use information gain to determine which features are more important than others. Conversely, filter approaches use an external performance measure to discriminate attributes, such as calculating their correlation with the target variable and with each other. Wrapper approaches use the actual data mining algorithm to determine which attribute combinations produce the best performance.

6.3.3.4 Feature Extraction

If any domain-specific knowledge is available, features that better represent the data can be extracted from the raw data and used in the modeling phase. This process is known as *feature extraction*. For financial time series, the technical indicators, such as MACD, are good examples. In economic data, fundamental indicators, such as consumer purchase index (CPI) are more appropriate. In biomedical data, the spectral centroid and spectral edge frequency might be used as features instead.

6.3.3.5 Mapping Data to a New Space

One common method to extract features from the data and reduce its dimensionality, is to use the statistical method of *principal component analysis (PCA)*. Principal component analysis transforms the data into a new space by redefining its dimensions. The basic concept is to figure out in what direction (dimension) the data has the highest variance. This dimension becomes the first feature of the dataset. The next step is to choose another direction (dimension), perpendicular to the first dimension, with the next highest variance. This becomes the second feature of the dataset. This process then continues until the desired amount of features has been obtained. Principal component analysis can, thus, also be used as a feature subset selection method.

Other forms of data transformations are available, such as *independent component analysis (ICA)* and switching between the time domain and the frequency domain using the *Fourier transform* and the *inverse Fourier transform* respectively (see example above).

6.3.4 Integrating the Data

Integrating the data includes data fusion and data aggregation. By fusing or aggregating data, the size of the dataset is reduced, which in turn speeds up the processing speed for data mining algorithms and reduces data storage.

6.3.4.1 Data Fusion

Data fusion was described above as the process of combining data from multiple, disparate sources. This step can either reduce the size of the resulting dataset if the individual datasets contain redundant information, or increase the size of the dataset if the information stored in the independent datasets is disjoint. As mentioned earlier, data fusion is a laborious task involving issues with duplicates, missing values and outliers.

6.3.4.2 Aggregation

Aggregation is the process of combining multiple rows into one. It effectively reduces the size of the dataset. An additional purpose of data aggregation is to change the resolution, or level of detail, of the dataset. For example, financial time series is commonly aggregated into intra-day (minute, hourly), daily, weekly, monthly and yearly data. An intraday trader is more interested in a dataset with an intra-day resolution, whereas a position trader is more interested in datasets with lower resolution. Another good example of aggregation is the roll-up and roll-down features provided in *online analytical processing (OLAP)*.

6.3.5 Formatting the Data

As a final task before the modeling phase, the dataset is formatted in various ways. Formatting the data does not change its meaning, but is intended as a step to produce compatible datasets for various data mining algorithms. For example, some data mining algorithms perform better if the order of the records in the dataset are randomized or if the data is normalized. Other algorithms might need the dataset to be partitioned into multiple datasets, that any class imbalance problem in the dataset is rectified, or that continuous attributes are discretisized since they cannot handle continuous attributes. If data mining tools (software products) are used, they might require the data attributes to be ordered in a certain way, such as starting with an initial identity attribute and ending with a final class attribute. Other tools might require records to be grouped according to class affinity.

6.3.5.1 Sampling

Sampling is a technique used to either reduce the size of a dataset, to change the composition of a dataset or to produce multiple dataset from one dataset. Reducing the size of a dataset is accomplished by sampling a certain amount of records from the initial dataset to comprise the smaller dataset. This can be done by sampling records using a constant interval or by random sampling.

Three random sampling techniques are usually used to change the composition of a dataset; *sampling with replacement (bootstrapping)*, *sampling without replacement* and *stratified sampling*. The difference between sampling with or without replacement is that in sampling with replacement the same record can be selected multiple times, whereas in

sampling without replacement a specific record can only be selected once. These two sampling techniques are commonly used to produce diversified datasets for classifier *ensembles* using the *bagging* and *boosting* algorithms (described in the modeling chapter).

Some data mining algorithms perform better if there is an even amount of records in the dataset with different class affinities. Stratified sampling is used to create this balance by randomly sampling (with replacement) an equal amount of records belonging to different classes. A good example is the SMOTE algorithm [19], which oversamples minority classes.

Sampling can also be used to partition the dataset into multiple datasets, as described above. Another reason for creating multiple datasets is to partition the datasets into training, validation and test sets (as described in the modeling chapter). The dataset partitions can either be created using random sampling or simply by slitting the dataset at certain points.

6.3.5.2 Discretization

Some data mining algorithms, such as decision trees, are designed to operate on discrete or categorical attributes. In such cases, a dataset with continuous attributes needs to be converted into a discrete dataset via a process known as *discretization*. One common way to discretize a continuous attribute is to divide its range into bins of equal width or frequency, for example, the three bins (-∞, 10], (10, 20], (20,∞) of equal range would produce a ternary attribute. *Binarization* is a special case of discretization where binary attributes are created consisting of two bins.

A nominal or ordinal attribute with N values can also be turned into a binary equivalent by assigning each value a unique number in the range $[0, N-1]$ and then converting them into $[[\log_2(N)]]$ binary attributes (the notation [[x]] means the highest integer). When converting an ordinal attribute using this scheme, order needs to be preserved.

6.3.5.3 Normalization

A handful of data mining algorithms that operate on numeric attributes are sensitive to small changes in real numbers, such as neural networks. This also applies to algorithms relying on measures of dissimilarity (distance measures), such as k-means and k-nearest neighbor, which are also sensitive to relative magnitudes of values between attributes. One common measure of dissimilarity is the Euclidean distance shown in Equation 6.1, where N is the number of attributes and x_k and y_k are the attribute values of two different objects (records).

$$d(x, y) = \sqrt{\sum_{k=1}^{N}(x_k - y_k)^2} \tag{6.1}$$

In order to deal with such issues, each attribute is usually normalized (or standardized). The simplest way to normalize an attribute is to subtract its smallest value x_{min}, from each value x_i, and then divide the result with the difference between its highest and smallest value x_{max}-x_{min} (i.e. the range of the attribute), as shown in Equation 6.2. This produces values in the range [0,1].

$$x_i' = \frac{(x_i - x_{min})}{(x_{max} - x_{min})} \tag{6.2}$$

A more common approach is to normalize each attribute by subtracting the attribute's sample mean μ, from each value x_i, and then dividing the result with the attribute's sample standard deviation s, as shown in Equation 6.3.

$$x_i' = \frac{(x_i - \mu)}{s} \tag{6.3}$$

The sample mean μ, is simply calculated by adding up all values x_i, and dividing by the total number of values N (Equation 6.4). The sample variance s^2, is calculated using Equation 5.5, from which the sample standard deviation s, can be obtained (Equation 6.6).

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i \tag{6.4}$$

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \mu)^2 \tag{6.5}$$

$$s = \sqrt{s^2} \tag{6.6}$$

6.4 Modeling

The modeling phase consists of selecting the modeling techniques (data mining algorithms) to be used to solve the data mining problem [20]. Furthermore, a test design needs to be created in order to test the quality and validity of the chosen modeling techniques. When this has been done, the actual models are built and assessed.

6.4.1 Selecting the Modeling Technique

There are plenty of modeling techniques available to choose from. Although, some techniques are more suitable than others depending on the goals of the data mining project and the nature of the dataset. Modeling techniques can be divided into two major categories; *descriptive modeling* and *predictive modeling*.

6.4.1.1 Descriptive Modeling

A descriptive modeling technique is used to extract hidden patterns from the data and to describe any discovered relationships. It is explorative in nature. Cluster analysis and association analysis techniques belong to this category. Cluster analysis techniques create clusters of data points (records) in the dataset where data points in the same cluster are more similar to each other than data points in other clusters. By examining the data points in each cluster, together with their attribute values, important relationships can be discovered. Four common clustering algorithms are *DBScan*, *k-means*, *hierarchical clustering* and *self-organizing maps (SOM)*. Association analysis techniques try to find associations between attributes by forming association rules from frequent item sets using an antecedent / consequent scheme. One good example is *market basket analysis*, commonly used to discover customer purchasing patterns in retail stores.

The other main modeling category, which is the main focus in this paper, is predictive modeling.

6.4.1.2 Predictive Modeling

A predictive modeling technique is used to predict the value of one attribute, the *dependent attribute*, based on the value of the remaining attributes, the *independent attributes*. If the dependent attribute is continuous, the predictive modeling task is known as *regression* and if the dependent attribute is discrete, it is called *classification*. Many data mining algorithms can handle both continuous and discrete attributes, hence they can be designed as either regression or classification models. Two good examples are *neural networks* and *hierarchical temporal memory*, which are described in chapters 7 and 8 respectively. Other common predictive modeling techniques are *k-nearest neighbors*, *decision trees*, *Bayesian networks* and *support vector machines (SVM)*.

This paper focuses on creating predictive classification models. The task of a classification model is to classify data points (records) into one of several classes. When creating, or training, a classification model, the class labels are usually stored in the dependent attribute. This type of training is called *supervised training* since the model is told which class a specific data point belongs to. When the model is tested, or deployed into a live environment, the trained model's task is to use the values of the independent attributes to estimate which class a specific data point belongs to. Some data mining algorithms can be trained without supplying known class labels in the dependent attribute, known as *unsupervised training* or *unsupervised learning*. Clustering is, in essence, unsupervised learning. During training, a classifier learns how to map independent attribute values to dependent attribute values by finding a decision boundary between the different classes. As an example, Figure 6.6 shows a linear decision boundary for a two-class problem. In the figure, the blue data points belong to one of the classes and the red data points to the other. In this case, the decision boundary does a good job at separating the two classes. Although, in most real-world applications, decision boundaries are more complex and non-linear.

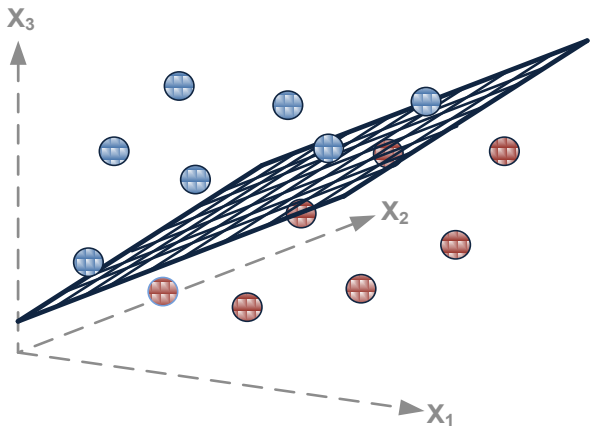

Figure 6.6 Decision boundary for a 2-class problem with 3 independent attributes.

6.4.2 Generating the Test Design

Once the modeling techniques have been selected, the next step is to devise a plan for training and testing the models. Common practice, when training classification models with only one dataset available, is to divide the dataset into either; a training- and a test set, or a training-, a validation- and a test set. In both cases, the models are trained using the training set and tested using the test set. The benefit of using a validation set, is that it can be used to "guide" the training of the models, or more precisely, to "guide" the search for an optimal model in the solution space.

One important property of a classification model is how well it generalizes to unseen data, i.e. how well it predicts the class labels of records it has never seen before. Therefore, it is important that a model not only have a low training error (the error rate on the training set), but also a low generalization error (the estimated error rate on previously unseen data). A model that fits the training set too well might have a worse generalization error than a model with a higher training error. This is called *overfitting*.

When dividing the dataset among the training-, validation- and test sets, bias is introduced (unless some sort of stratified sampling is employed). One successful remedy is to use *k-fold cross validation*, in which the data set is divided into k equal bins. Then $k-1$ of the bins are used to train a model and 1 to validate it. This is repeated k times so that each of the k bins is used once, and only once, as the validation set. Finally, the error rate over all runs is averaged to produce an unbiased measure.

6.4.2.1 Performance Measures

For classification problems, the models are evaluated using a confusion matrix. A confusion matrix is shown in Figure 6.7 for a binary classification problem. The confusion matrix shows if one class is being confused with another. In the figure, the true positives (TP) count is the number of classifications the network made of the positive (+) class when the actual class was the positive class. Similarly, the true negatives (TN) indicate that the classifier managed to guess the negative (-) class right when the actual class really was the negative class. The false negatives (FN) and false positives (FP) show how many times the classifier confused one of the classes with the other; predicting the negative class when the true class was the positive class and predicting the positive class when the true class was the negative class respectively.

		Predicted Class	
		Class +	**Class -**
Actual Class	**Class +**	TP	FN
	Class -	FP	TN

Figure 6.7 Confusion matrix for a two class problem.

From the confusion matrix, a number of performance measures can be calculated. The error rate (Equation 6.7) shows how many incorrect classifications were made relative to the total number of classification attempts, whereas the accuracy (Equation 6.8) shows how many correct classifications were made relative to the total number of classification attempts. The precision (Equation 6.9) shows how many true positive classifications were made relative to all positive classifications. The recall, also known as sensitivity (Equation 6.10), is the fraction of positive examples correctly predicted by the model. The specificity (Equation 6.11) shows the fraction of negative examples correctly predicted by the model. The F_1 measure determines the harmonic mean between the recall and precision (Equation 6.12), where a high F_1 value indicates that both precision and recall are fairly high.

$$\text{error rate} = \frac{FN+FP}{TP+FN+FP+TN} \tag{6.7}$$

$$\text{accuracy} = \frac{TP+TN}{TP+FN+FP+TN} \tag{6.8}$$

$$\text{precision} = \frac{TP}{TP+FP} \tag{6.9}$$

$$\text{recall (sensitivity)} = \frac{TP}{TP+FN} \tag{6.10}$$

$$\text{specificity} = \frac{TN}{TN+FP} \tag{6.11}$$

$$F_1 = \frac{2}{\frac{1}{r}+\frac{1}{p}} = \frac{2*\text{recall}*\text{precision}}{\text{recall}+\text{precision}} = \frac{2*TP}{2*TP*FP*FN} \tag{6.12}$$

Another commonly used performance measure is the *Area Under the ROC Curve (AUC)*, where ROC stands for *Receiver Operating Characteristics*. Figure 6.8 displays two ROC curves, where the blue curve represents a model with good performance and the red curve a model with poor performance. In fact, a straight diagonal line represents a model that performs random guessing, the worst performance possible. As can be seen in the figure, the closer the curve is to the upper left corner, the larger the area under the curve, and hence, the better the performance.

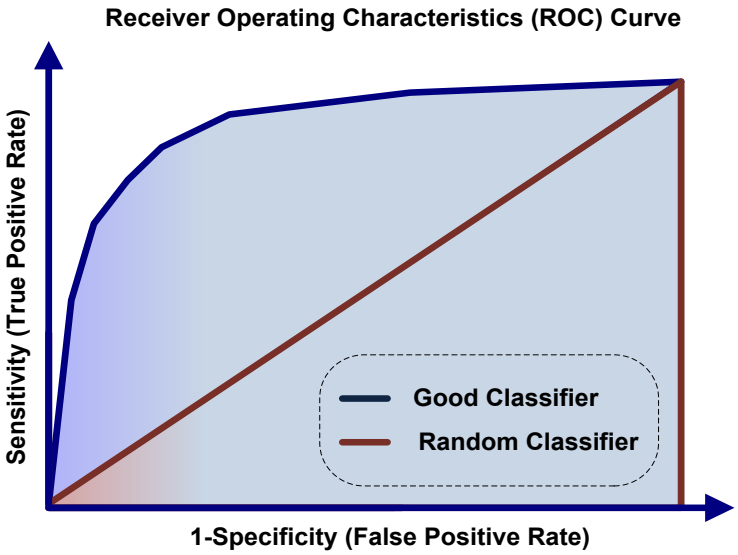

Figure 6.8 ROC curve for a good classifier (blue) and for a random classifier (red).

6.4.2.2 Ensembles

The purpose of ensemble methods is to increase model performance by training multiple base models on the training set and aggregating their individual predictions. For regression models, the outputs from all base models are simply averaged and used as the ensemble's value. For classification models, a majority voting scheme is used to determine the output from the ensemble. In order for ensemble methods to work, the base models should be independent of each other and should have a performance better than random guessing. Ensembles can be created by manipulating the training set, the input features or the learning algorithm.

Manipulating the training set is usually done by randomly sampling from the original training set multiple times in order to create one different subset for each base model. Bagging [21] and Boosting [22] are two common algorithms for creating ensembles by manipulating the training set. Bagging uses bootstrapping to create the base models' subsets, each the same size as the original training set. Boosting uses a weighting scheme to iteratively change the distribution of training samples so that the base classifiers concentrate on samples that are hard to classify.

Manipulating the input features is done by selecting a different subset of features for each base model. Random forests, a decision tree ensemble technique, uses this approach.

Manipulating the learning algorithm is applicable to data mining algorithms who's parameters can be changed in order to produce diverse models. One good example is neural networks, where the number of hidden layers, number of nodes in each layer, the learning algorithm and type of activation functions can be modified.

6.4.3 Building the Models

When the modeling techniques have been chosen and a testing plan has been devised, the models are built by exposing them to the training and validation datasets.

6.4.4 Assessing the Models

During this task, the models are assessed and the success of the data mining attempt determined. For a descriptive data mining approach, the models are assessed based on their success of discovering interesting patterns in the data. Predictive models are assessed as to how successful they were in estimating the value of the dependent attribute. The overall test design is also assessed as part of this task.

6.5 Evaluation

The evaluation phase in CRISP-DM consists of the three tasks; evaluating results, reviewing the data mining process and determining the next steps. These task are more business focused and are out of scope for this paper.

6.6 Deployment

The final CRISP-DM phase consists of the four tasks; planning deployment, planning monitoring and maintenance, producing a final report and reviewing the project. These task are also out of scope for this paper.

This chapter described the general data mining process. The next three chapters describe the machine learning technologies used in this paper to create predictive models of financial markets; neural networks, hierarchical temporal memory and evolutionary learning respectively.

7 Neural Networks

Neural networks are based on the idea of interconnecting neurons in the biological nervous system. This chapter gives a short introduction to the biological equivalent of interconnected neurons followed by a detailed description of neural networks.

7.1 The Biological System

All content in this subchapter is adopted from [23].

In the biological system a generalized neuron consists of a cell body (*soma*) with small tentacle-like protrusions (*dendrites*) extruding from one end of the neuronal soma. In the other end, a long tail-like protrusion forms a structure called an *axon*, which connects to the soma at a point called the *axon hillock*. At the very end of an axon, a structure called a *synaptic knob* (or *axon terminal*) is formed. Axons divide into multiple branches to form multiple knobs at their very ends.

Neurons connect to each other by attaching their terminal ends to other neurons' dendrites, soma or axon hillocks. The connection is formed indirectly through a small gap, called the *synaptic cleft*. A neuron connecting its axon to another neuron's dendrite is called a *presynaptic neuron*, and the latter neuron is called the *postsynaptic neuron*. Presynaptic neurons send bioelectrical signals (*action potentials*) down their axons which are transferred

to postsynaptic neurons via the synaptic cleft. When the action potential reaches an axon terminal, it opens up voltage-gated, *transmembrane protein channels* which cause an influx of calcium ions (Ca^{2+}) into the synaptic knob. This, in turn, causes sack-like structures (*vesicles*) within the knob to migrate and merge with the cell membrane, whereby they release their content (*neurotransmitters*) into the synaptic cleft via a process called *exocytosis*. The neurotransmitter travels across the synaptic cleft and attaches to receptors connected to chemically-gated, transmembrane protein channels, which in turn causes an influx of sodium ions (Na^{+}), and efflux of potassium ions (K^{+}), across the postsynaptic neuron's cell membrane. This eventually builds up a graded potential across the neuronal membrane until it reaches threshold potential, which then causes an action potential to be fired and sent down the postsynaptic neuron's axon. This process is repeated from neuron to neuron.

Action potentials can either have an inhibitory or excitatory effect on the postsynaptic neuron's membrane. The resulting postsynaptic bioelectrical signal is therefore known as either an *inhibitory postsynaptic potential* (*IPSP*) or a *excitatory postsynaptic potential* (*EPSP*). When all postsynaptic signals are summed-up, including signs, the *grand postsynaptic potential* (*GPSP*) is formed. This is the graded potential that causes an action potential to be fired once threshold potential has been reached. Neurons interconnect, using the scheme described above, to form vast hierarchical, electrochemical communication structures within the nervous system.

Similarly, in the technical system, a neuron has weighted (inhibitory or excitatory) inputs (dendrites) and an output (axon). The weighted inputs are summed together (forming the GPSP), which causes an output signal (action potential) to be generated as determined by the activation function (when threshold potential is reached an action potential is fired). The output from one neuron becomes the input to another neuron. A simple neuron in the technical system is called a *perceptron*.

7.2 The Perceptron

In 1943, McCulloch and Pitts created the first mathematical model of a biological neuron, called the perceptron [24]. It had all the basic capabilities of a biological neuron such as; a set of weighted inputs (synapses), a summation unit (for adding up the weighted inputs, hence building up the graded potential) and an activation function (in the form of a simple step function which determined when an action potential was fired). The perceptron is illustrated in Figure 7.1.

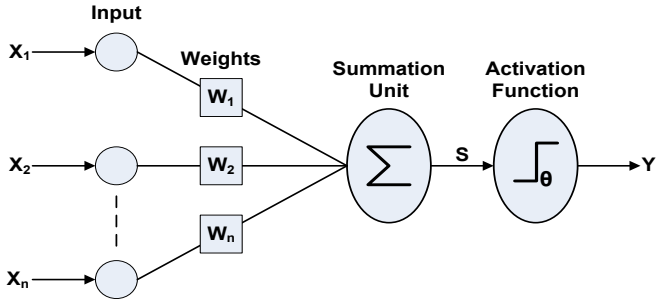

Figure 7.1 The Perceptron

To calculate the output from this simple perceptron, the summation unit adds up its weighted inputs, $\{w_1x_1, w_2x_2, \ldots, w_nx_n\}$ according to Equation 7.1 to produce the sum S, followed by passing the sum through the activation function which outputs a $'1'$ if the threshold has been reached or $'0'$ otherwise according to Equation 7.2.

$$s = \sum_{i=1}^{n} w_i x_i \tag{7.1}$$

$$y = f_\theta(s) = \begin{cases} 1, if\ s > \theta \\ 0, if\ s \leq \theta \end{cases} \tag{7.2}$$

One and a half decades later, Rosenblatt refined the perceptron model and build the first, single-layer perceptron network [25]. The network consisted of input nodes, which simply received one input each and forwarded them to the single layer of perceptrons. As usual, each input is multiplied with its corresponding weight and the weighted sum of all inputs are added together by the perceptron before passing through its activation function. A single-layer perceptron network is shown in Figure 7.2, where each perceptron in the output layer receives weighted inputs from all input nodes.

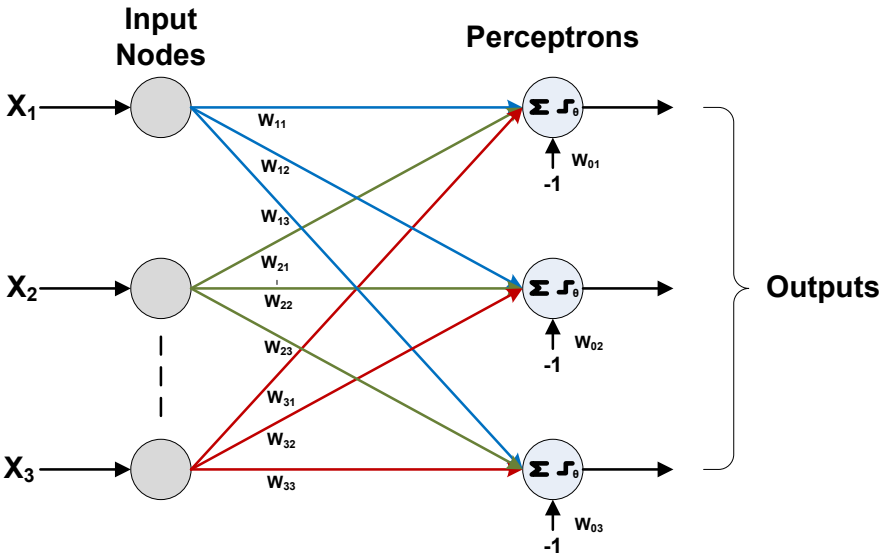

Figure 7.2 The Single-Layer Perceptron Network

Rosenblatt also added a constant bias term with a corresponding weight to each perceptron. The bias term is shown as *-1* in the figure above. The reason for doing this was to enable the perceptron to adjust its idle threshold value. This permitted each individual perceptron to configure its own activation level.

Furthermore, a simple update rule was devised so that the perceptron could update its weights w_{ij}, in order to produce a target output t_j, for a given input x_i. Although, to prevent the weights from adjusting their values too quickly, a learning rate parameter η, was introduced. The complete weight update rule is shown in Equation 7.3, where the true output is y_i. The complete output calculation for a perceptron is reiterated in Equation 7.4.

$$w_{ij} \leftarrow w_{ij} + \eta(t_j - y_i)x_i \tag{7.3}$$

$$y_i = f_\theta\left(\sum_{i=1}^{n} w_{ij}x_i\right) = \begin{cases} 1, if\ w_{ij}x_i > \theta \\ 0, if\ w_{ij}x_i \leq\ \theta \end{cases} \tag{7.4}$$

Using the two equations above, the perceptron learning algorithm can be defined as:

1. Initialize all weights to small random, positive and negative numbers.
2. For k iterations
    a. Repeat for all input vectors ***X***
        i. Use activation function f_θ to calculate the activation for each perceptron *j* (Equation 7.4).
        ii. Update each weight using Equation 7.3.

Once the perceptron has been trained, Equation 7.4 can be used to calculate its output for any input. Rosenblatt used the algorithm to train a perceptron to solve the OR problem, by setting the activation threshold to *0* and the bias input to *-1*. The inputs and target outputs are shown in Table 7.1 and the perceptron with its two inputs in Figure 7.3. From the table it can be seen that the output should be *1* when either of the inputs are *1*.

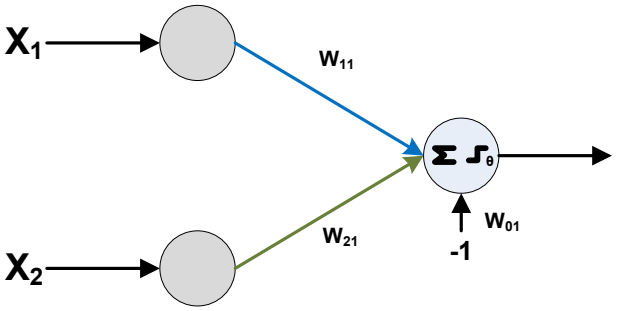

Table 7.1 OR Truth Table

x_1	x_2	t
0	0	0
0	1	1
1	0	1
1	1	1

Figure 7.3 The OR Perceptron

This simple perceptron, including single-layer perceptrons in general, create a linear decision boundary between different output targets. This is shown in Figure 7.4 (left) for the OR problem, where the red data point symbolizes the data point for the two zero inputs, and the blue data points the other three combinations. The black boundary line does a good job at separating the blue data points from the red. If the problem is turned into an XOR problem, by substituting the bottom target value in Table 7.1 with a *0*, the perceptron fails to find a solution to the problem, since it cannot find a linear decision boundary between the blue and red data points anymore, as shown in Figure 7.4 (right). This shortcoming with the single-layer perceptron was proven in a paper by Minsky and Papert in 1969 [26], which caused any further research into neural networks to be put on ice for the next twenty years.

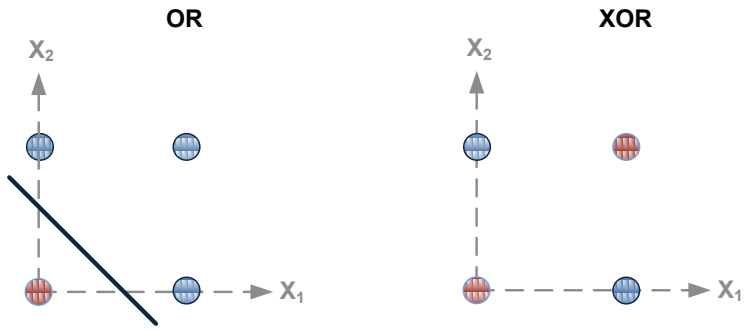

Figure 7.4 Input space for the OR problem (left) and the XOR problem (right).

7.3 The Multi-Layer Perceptron (MLP)

Two decades later, neural network research was given a kick in the right direction again when the *Multi-Layer Perceptron* (*MLP*) network was invented, together with the first practical neural network learning algorithm, the *backpropagation* algorithm [27], [28]. In 1986, Rumelhart, Hinton and McClelland demonstrated that their MLP could solve non-linear problems by adding additional layers of perceptrons, introducing non-linear activation functions and back-propagating the output error through the entire network in order to adjust the network weights. They demonstrated this by solving the XOR problem that had haunted the research field for twenty years, as shown in Figure 7.5.

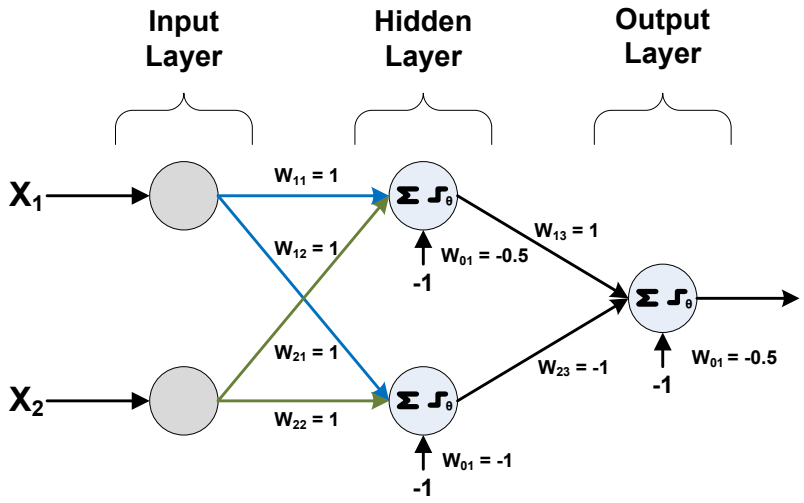

Figure 7.5 The XOR MLP Network

MLPs are created in a hierarchical structure with perceptrons as its building blocks, where multiple perceptrons are grouped together into layers. The initial layer is called the input layer, followed by one or more hidden layers, and a final output layer. Each perceptron, in each layer, receives inputs from perceptrons in lower layers and output signals to higher layers. Each perceptron calculates the weighted sum of its inputs, including a bias input, and sends the resulting signal through its activation function, as usual. The activation function then decides what the output of the perceptron should be. Input vectors are supplied to the lowest layer (input layer), and the output from the top layer (output layer) is the output from the MLP network.

Running the MLP in the forward direction is not much different from running the single-layer perceptron network above. Although, the calculation of the output error for the single-layer perceptron was a bit naïve, i.e. the true output value was just subtracted from the target value. Using such an approach, two subsequent errors with the same magnitude, but different signs would cancel each other out, effectively hiding two errors. A better error function would be to use the sum of squared errors as in Equation 7.5. By adding up the squared errors of all the output nodes, all errors will be accounted for. The additional division by 2 before the summation symbol is just a convenience measure for calculating the derivative of the error function, which is mandatory for the backpropagation algorithm as explained in the text below.

$$error = \frac{1}{2}\sum_{i=1}^{n}(t_i - y_i)^2 \tag{7.5}$$

Another modification to the algorithm in the forward direction, as compared to the single-layer perceptron, is a slight modification of the activation function. The backpropagation algorithm also requires the activation function to be differentiable. The single-layer perceptron, above, used a step function which has a discontinuity at 0 (at the "step"). Two continuous functions that look a lot like the step function are the *sigmoid function* (also known as the *logistic function*) and the *hyperbolic tangent function* (also known as the *symmetric sigmoid function*). The sigmoid function (Equation 7.6) ranges from 0 to 1, has a near linear, middle range, and saturates at its boundaries. The hyperbolic tangent function (Equation 7.7) has the same properties as the sigmoid function, but ranges from -1 to 1. For the following discussion, the sigmoid function will be used (the β term is just a scaling factor and can be set to 1).

$$sig(x) = \frac{1}{1+e^{-\beta x}} \tag{7.6}$$

$$\tanh(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}} \tag{7.7}$$

For the single-layer perceptron, the simple weight update rule (Equation 7.3) sufficed, since all inputs and outputs were known to each perceptron. The initial problem that faced Rumelhart, Hinton and McClelland in designing the backpropagation algorithm was that the output layer did not know about the inputs, the first hidden layer did not know about the outputs and any additional hidden layers (between the first hidden layer and the output layer) did not know about either the inputs or the outputs. Therefore, a more sophisticated mechanism was needed in order to propagate the output errors back through the network to update all network weights. The solution was to differentiate the error function and the activation functions with respect to the weights and then run the network backwards with the error now acting as the input from the output layer. This enabled the output error to be propagated backwards to calculate the errors in the hidden layers, which in turn enabled the weights to be adjusted to minimize all errors. In order to do this, the backpropagation algorithm uses a gradient descent mechanism based on the partial derivatives of the error function with respect to the weights. By taking the negatives of the partial derivatives, the direction of the error with respect to each weight is obtained. Hence, by updating the weights in the opposite direction of the error, the error caused by each weight is minimized.

Running a MLP using backpropagation in the forward direction is not much different from the single-layer perceptron, i.e. the calculations for each perceptron (layer by layer) is identical to Equation 7.2 with the exception that the activation function f_θ, is replaced by the sigmoid function (Equation 7.6) in this case. The learning algorithm for a MLP with one hidden layer, sigmoid activation functions and using backpropagation is defined as:

1. Initialize all weights to small random, positive and negative numbers.
2. For k iterations (or until the error on the validation set increases)
    a. Repeat for all input vectors $\boldsymbol{X}$
        i. Calculate the output of each perceptron, h, in each hidden layer

$$s_h = \sum_i x_i w_{ih} \tag{7.8}$$

$$a_h = f_\theta(s_h) = \frac{1}{1+e^{-\beta s_h}} \tag{7.9}$$

        ii. Calculate the output of each perceptron, o, in the output layer

$$s_o = \sum_h a_h w_{ho} \tag{7.10}$$

$$a_o = f_\theta(s_o) = \frac{1}{1+e^{-\beta s_o}} \tag{7.11}$$

        iii. Calculate the error for each perceptron, o, in the output layer

$$\delta_o = (t_o - a_o) a_o (1 - a_o) \tag{7.12}$$

        iv. Calculate the error for each perceptron, h, in each hidden layer

$$\varepsilon_h = a_h (1 - a_h) \sum_o w_{ho} \delta_o \tag{7.13}$$

        v. Update the weights in the output layer

$$w_{ho} \leftarrow w_{ho} + \eta \delta_o a_h \tag{7.14}$$

        vi. Update the weights in each hidden layer

$$w_{ih} \leftarrow w_{ih} + \eta \varepsilon_h x_i \tag{7.15}$$

The learning algorithm described above is sequential since it updates the weights for every input. Another way of running the algorithm is in batch learning mode, in which all training samples are fed through the network before the weights are updated as a final step. Batch learning produces a more accurate estimate of the error gradient and therefore converges to the local minimum more quickly. Because the backpropagation algorithm, or more precisely the gradient descent algorithm, converges on local minima, a momentum term can be added to the two final steps in the algorithm (Equations 7.14 and

7.15). The momentum term adds in a fraction of the previous weight update to the current update so that the algorithm has a chance to escape a local minimum. The modified equations including the momentum term, α, are shown in Equations 7.16 and 7.17. The learning rate parameter, η, can also be reduced gradually during training to expedite convergence.

$$w_{ho}^{t} \leftarrow w_{ho}^{t} + \eta\delta_o a_h + \alpha\Delta w_{ho}^{t-1} \tag{7.16}$$

$$w_{ih}^{t} \leftarrow w_{ih}^{t} + \eta\varepsilon_h x_i + \alpha\Delta w_{ih}^{t-1} \tag{7.17}$$

One rule of thumb is to initialize all weights randomly within the interval $\pm 1/\sqrt{n}$, where n is the number of input nodes. This causes an initial sum into a perceptron of roughly 1. This initial setting increases the chances of all weights reaching their final values at the same time, commonly known as *uniform learning*. Since the sigmoid activation function saturates at 0 and 1, it is highly recommended to normalize, or standardize, all inputs before feeding them to the network. The most common way to normalize the inputs is to use Equation 6.18, where μ is the mean for one input (dimension) and σ is the standard deviation. Another rule of thumb is to use ten times as many training samples as there are network weights. This is due to the sparsity in weight space that arises due to the curse of dimensionality. On the other hand, no rule of thumb is available when it comes to choosing the number of nodes to use or the number of hidden layers. This is usually determined experimentally.

$$x_i' = \frac{x_i - \mu}{\delta} \tag{7.18}$$

MLPs can be used in classification and regression. When the MLP is used in regression, the output nodes have linear activation functions, i.e. the activation function is simply $f_\theta(x) = x$. The evaluation function for the MLP is then usually calculated as the sum of squared errors (Equation 7.5 but without the division by 2 before the summation symbol). When used as a classifier, the MLP can either use sigmoid activation functions in its output layer (Equation 7.6) or the *softmax function* (Equation 7.19). The softmax function rescales the outputs by calculating the exponential of the inputs to a perceptron and dividing by the total sum of the inputs to perceptrons in the same layer. This way the activations sum to 1 and range from 0 to 1. The softmax activation function is usually used for classification problems that use *1-of-N encoding*, i.e. instead of using one output perceptron for, e.g. a ternary classification problem, three perceptrons are used. Then, the target vector is turned into a target matrix, in this case with three columns (one for each class), where a 1 is indicated for the actual class in one of the three columns and a 0 in the other two. This is done for each row in the target matrix.

$$a_o = f_\theta(x) = \frac{e^x}{\sum_{\hat{x}} e^{\hat{x}}} \tag{7.18}$$

For classification problems, the MLPs are evaluated using a confusion matrix as described in chapter 6.

When training a neural network, a validation set is commonly used to enable an early stopping criterion for the backpropagation algorithm and to prevent the network from overfitting the data. After a number of epochs (an epoch is defined as running all training samples through the network once), the network is validated on the validation set to measure its performance. At first, the network will show poor performance on the validation set since it has not been through enough training epochs. As training continues, the networks performance on the validation set will gradually increase, up to a point, where it will turn around and start getting poorer again. This is a good signal for stopping training, since it implies that the network has started overfitting the training set.

Other forms of learning algorithms have been introduced over the years, such as the conjugate gradient algorithm [29] and the RProp algorithm [30]. These are newer and quicker versions of the backpropagation algorithm. Other neural network derivatives have also seen the day of light over the years such as recurrent networks, radial basis function (RBF) networks and an unsupervised version known as self organizing maps (SOMs). Recurrent networks are specifically suitable for sequential data such as financial time series. In a recurrent network, output signals from hidden layers are fed back as inputs to lower layers. This contraption enables the recurrent network to learn about serial dependencies in the time series. In fact, recurrent networks are used as the benchmark technology in this study. A more recent technology called Hierarchical Temporal Memory, which is the topic of the next chapter, borrows ideas from neural networks, and combines them with ideas from Bayesian networks and spatiotemporal clustering algorithms.

8 Hierarchical Temporal Memory (HTM)

The efficiency and robustness of the human brain has captured the fascination of scientists for centuries. Its ability to extract and memorize patterns from vast amounts of sensory information, only to be effortlessly recalled and associated with novel input, is far superior to other species and unparalleled to contemporary, computerized systems. How is it that a child can learn to recognize an apple no matter what color, shape or size it is, independent of lighting conditions and despite of any deformations even though the apple might be partially concealed, when a super computer would struggle with the same recognition task? How is the mammalian brain able to memorize vast amounts of sensory patterns and later recall, adapt, and associate them to novel input patterns within a fraction of a second when a cluster of super computers with huge amounts of random access memory and central processing units find the chore almost infeasible? Would it be possible to create more efficient learning technologies if the current architectures were replaced by designs based on the structural and algorithmic properties of the human brain? These philosophical questions were the precursors to a subfield of machine learning called *deep machine learning* architectures [31].

Recent research within the field of neuroscience has provided valuable insight into how the mammalian brain encodes sensory information within hierarchically organized neuronal formations, called *microcolumns*, spanning the six layers of the *neocortex* [32]. The neocortex is the latest evolutionary addition to the central nervous system and is the outmost layer of the cerebrum, which surrounds the two hemispheres in the mammalian

brain. Not only does the hierarchical organization of these processing units, i.e. microcolumns, enable efficient learning of spatiotemporal patterns across different sensory modalities, e.g. visual, auditory and somatosensory, but also applies to the efferent (somatomotor) subdivision of the nervous system [33].

Mimicking their biological counterpart, deep machine learning technologies incorporate hierarchical information processing into their design. Some architectures model hierarchical structures in either the spatial- or temporal dimensions, others attempt to model both dimensions simultaneously [34]. One neocortical deep learning technology of the latter type, is Hierarchical Temporal Memory which is described in this chapter.

8.1 The Biological System

In 1957 Vernon Mountcastle discovered the columnar organization of neurons within the mammalian cerebral cortex [35]. At the time it was already known that the cortex was organized into six layers containing different types and densities of neurons. The novelty in Mountcastle's discovery was the vertical grouping of neurons across the six layers. The right picture in Figure 8.1 shows the structure of a microcolumn passing through the six layers in the cerebral cortex, with a coronal view of the brain in the middle. Layer I is closest to the scalp and layer VI is closest to the brain's white matter.

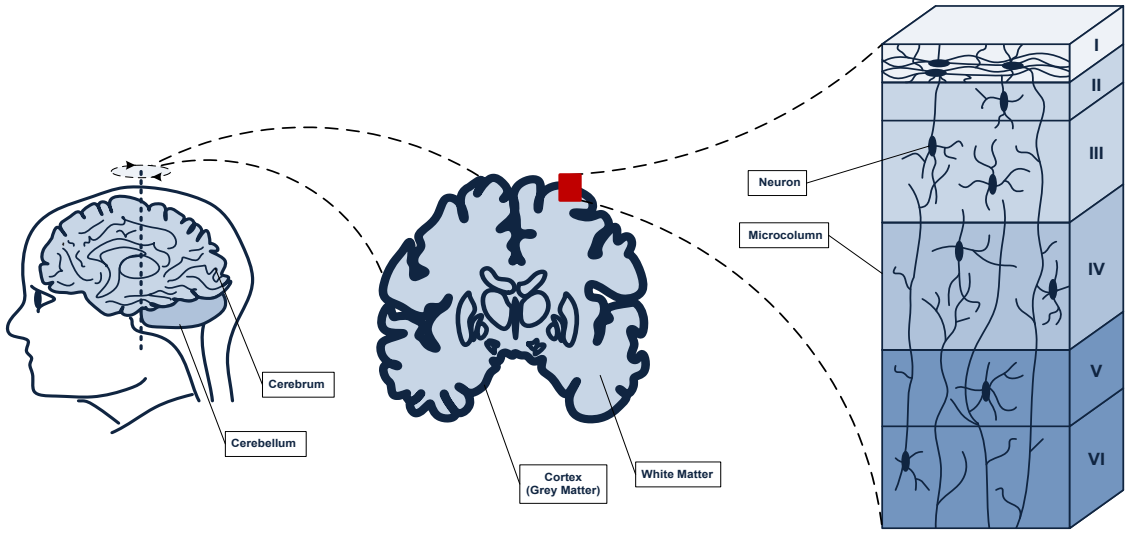

Figure 8.1 The Microcolumn structure in the cerebral cortex.

Each grouping, or microcolumn, consisted of about 100 neurons which acted together as a single processing unit. If, for example, a certain visual pattern was displayed on the retina of a cat's eye, the neurons within a certain microcolumn would become active at the same time. Mountcastle observed the same basic microcolumnar formations in different sensory modalities, i.e. in the visual-, auditory- and somatosensory cortex. He also noticed that the microcolums were organized in hierarchical, tree-like structures throughout different regions of the cortex. His theory of hierarchically distributed uniform microcolumnar processing units for cerebral function was presented in 1978 [32].

8.2 Memory Prediction Theory of Brain Function

Mountcastle's unit model of microcolumnar processing units captured the interest of Jeff Hawkins. Hawkins started his career in computer science and electrical engineering, working as a computer platform architect at the Intel corporation during the time the first microprocessors were created and later founded the Palm Computing corporation where he architected the Palm Pilot™ and Treo™ smart phone. Despite his lucrative engineering career, Hawkins had always had an obsession with the human brain. Therefore, having obtained his financial independence through his entrepreneurial success, Hawkins started studying biophysics at the University of California at Berkeley and later founded the Redwood Neuroscience Institute. His newfound knowledge within neuroscience coupled with his lengthy experience in designing mobile computing platforms, gave him an advantageous insight into how computer-based systems could benefit from the structural- and behavioral aspects of the human brain. Although Hawkins had multiple theories of how the brain worked and how this knowledge could be implemented in computer software and hardware, something was missing. It wasn't until Hawkins stumbled upon Mountcastle's theory of a common cortical processing unit, that he could complete his puzzle and publish his *memory prediction framework* of brain function in his book *On Intelligence - How a new understanding of the brain will lead to the creation of truly intelligent machines* in 2004 [33]. Hawkins' memory prediction model is based on a network of microcolumns, hierarchically connected to each other throughout different regions of the cortex, as illustrated in Figure 8.2. The top, middle picture shows an axial view of the brain, where three different parts of the cortex have been color-coded in red (V1), green (V2) and light blue (V4). Sensory information is processed by the red region first, which is forwarded to the green region and then the light blue region respectively (green arrows). Information is then passed back down to the lower regions again (red arrows). The bottom, middle picture shows the hierarchical structure of information processing between the different regions. The right picture shows the processing units organized into microcolumns, spanning the six layers of the cerebral cortex.

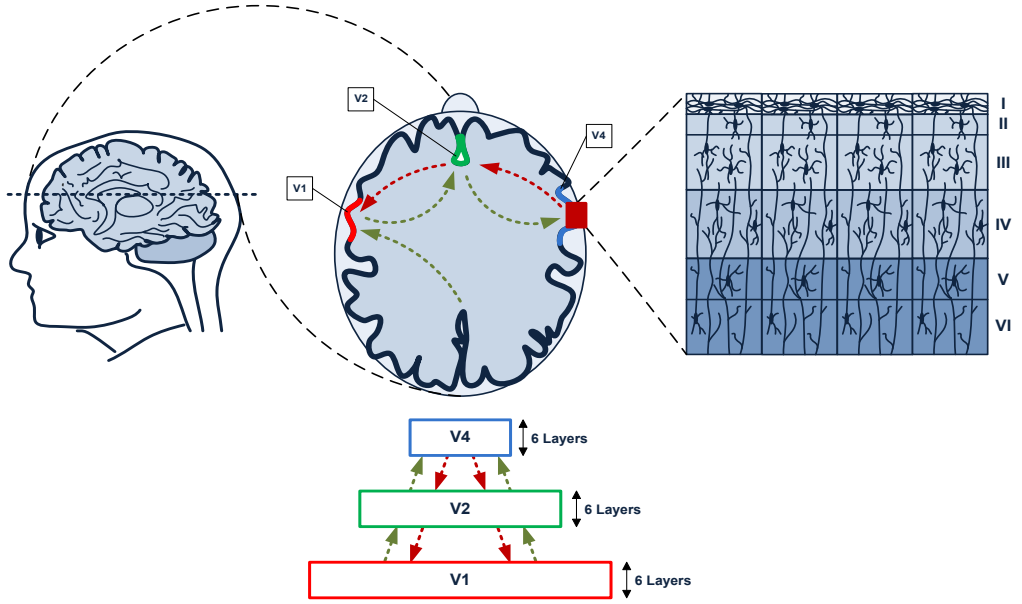

Figure 8.2 A hierarchical network of microcolumns in the cerebral cortex.

8.3 A Mathematical Model of Brain Function

Hawkins now had a theoretical model of brain anatomy and behavior that could potentially be implemented as a machine learning technology. In order to do this, however, he needed to create a mathematical model of his neocortical model. Dileep George, a PhD student in Electrical Engineering at Stanford University, came to his rescue. In 2005, he devised a mathematical model equivalent to Hawkin's biological model, based on a hierarchical Bayesian network, incorporating spatiotemporal clustering algorithms and concepts drawn from neural networks [36]. He also made use of Bayesian Belief Propagation to propagate beliefs between network nodes [37] (the mathematical equivalent to the red and green arrows in Figure 8.2). A detailed description of George's mathematical model was presented in his PhD thesis in 2008 [38]. With the theoretical and mathematical models in place, Hawkins and George cofounded the company Numenta Inc in 2005 and started working on their first version of the Hierarchical Temporal Memory (HTM) machine learning technology [39]. Numenta currently offer a free software version of their HTM development platform (NuPIC version 1.7.1) for research purposes, which is the software version used in this paper [40].

8.4 HTM Concepts and Terminology

Hierarchical Temporal Memory (HTM) is a machine learning technology based on Hawkins' memory prediction theory of brain function and implemented using George's mathematical model. Both structural and behavioral (algorithmic) properties of the neocortex are part of the model. The structural property, consisting of hierarchically organized processing units (microcolums) in the biological system, is implemented as hierarchically connected nodes in the technical system, equivalent to the structure of neural networks. The algorithmic property, based on the memory function of sensory information included in each microcolumn in the biological system, is implemented using concepts drawn from Bayesian networks, Bayesian belief propagation and clustering algorithms in the technical system.

Since the HTM technology is based on sensory information processing in the biological system its operation is defined as *discovering causes in the world* and *inferring causes of novel input* [41]. *The world* is nothing more than the inputs to the HTM network (compare with the inputs to a neural network). *Causes* can be thought of something that is *causing patterns* in the world (input). For example, a cause (object) in the world (input) that is causing a pattern of increasing distance and speed (two input features) might be an accelerating car. The accelerating car is causing these patterns currently sensed by the HTM network. Other causes of patterns in the world are conceptualized in the left picture in Figure 8.3. Sounds, words, temperatures, shapes and people are all examples of objects in the world that can cause patterns sensed by an HTM network. An HTM's sensory array consists of its inputs as shown in the middle of the figure. The sensory array senses the patterns which are fed into the HTM network. The HTM network then forms beliefs of which causes are currently present in the world, as shown to the right in the figure. For example, if the temperature is cause 2 in the figure, the HTM network currently believes, with a 72% probability, that the temperature is causing the patterns currently sensed by the HTM network.

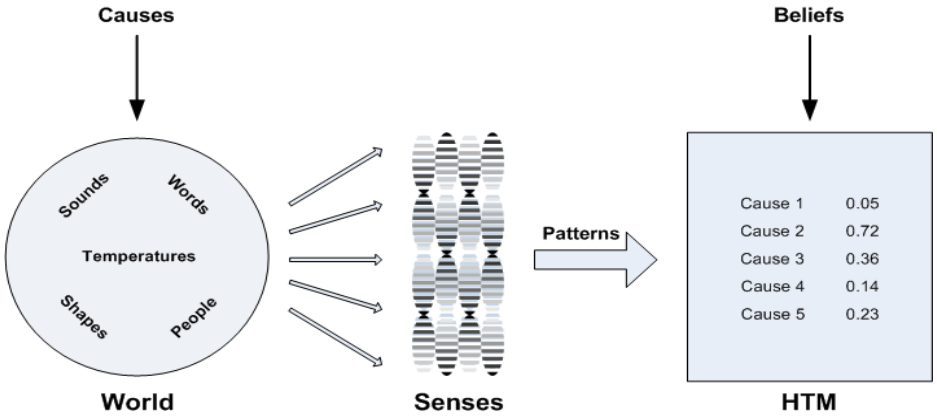

Figure 8.3 Discovering causes of patterns in the world.

There is a hierarchy of causes active in the world at any moment in time, for example letters, words and sentences, which are causes of sound sensed by the auditory sensory system in spoken language. Each input measures a local and simple quantity, such as the letters or temperature readings in the previous examples.

Once an HTM network has been trained to recognize causes of patterns in the world, it can infer causes from novel input. There is however two prerequisites for training an HTM model; *the causes in the world must persist over time*, and their *patterns must flow continuously through time*. In other words, the HTM technology requires a continuous, sequential dataset during training, such as a time series of temperature and air pressure measurements in a weather forecasting system. For financial time series, these prerequisites are fulfilled by the time-varying fluctuations in price and trade volume levels.

Figure 8.4 shows the basic structure of an HTM network consisting of four input nodes, two middle-layer nodes and one top node. The arrows (except the actual input stream) represent the memory function, or beliefs, calculated by each node and passed up the hierarchy. Each node performs the same learning algorithm in order to discover and learn causes in the world. Based on the current input sample, each node assigns a probability to each of its learned causes. If a node has learned 5 causes, it will therefore calculate a probability vector consisting of 5 elements, each containing a probability value for one of the learned causes (see Figure 8.3). This probability vector is a node's output, which becomes the input to a higher-level node. Lower-level nodes will learn about simple causes in the world, e.g. notes in a song. As the information ascends the hierarchy, higher-level nodes will learn more sophisticated causes in the world, e.g. sequences of notes in a song. The most sophisticated causes will be learned by top-level nodes, e.g. a complete song.

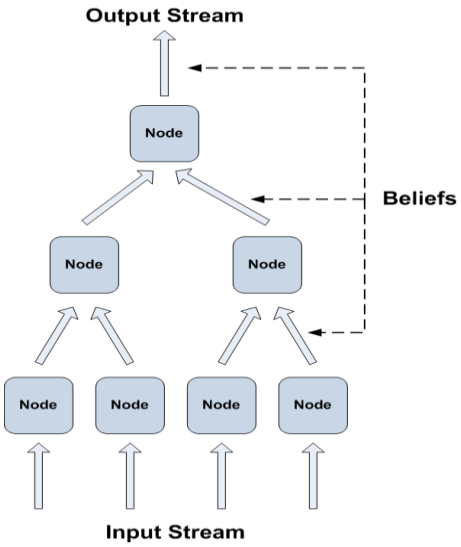

Figure 8.4 Structure of an HTM Network.

Each HTM node maintains two memory banks; one for patterns it has learned and the other for sequences of such patterns it has learned. For example, each individual note in a song is a pattern learned by low-level nodes. Sequences of such notes constitutes the sequences of patterns stored in the other memory bank. Using HTM terminology, patterns stored in the first memory bank are called *coincidences* and sequences of coincidences (patterns) are known as temporal *groups*. Therefore, coincidences store spatial information in the input stream and groups store temporal information. The first memory bank is maintained by something called a *spatial pooler*, since it pools (groups together) spatial patterns, and the second memory bank is maintained by a *temporal pooler*, since it pools temporal patterns. For a financial time series, the spatial patterns (coincidences) in lower-level nodes are made up of price and trade volume levels and the temporal patterns (groups) are made up of sequences of price and trade volume levels. Hence, coincidences in the same group are more likely to follow each other in time.

Sample spatial patterns for a bottom level node are displayed in Figure 8.5. In this case, the input to the node consists of a 16-bit, binary vector where each element can take on the value 0 or 1. If the patterns occur frequently, the node learns that these are common patterns and therefore remembers them. This is the case with the patterns displayed across the top row in the figure, where the patterns are (from left to right); a vertical line, a vertical line, a bottom left corner and a top right corner. Uncommon patterns, displayed along the bottom row, are ignored by the node.

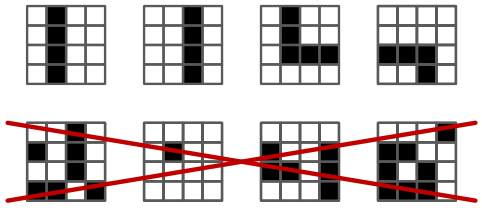

Figure 8.5 Spatial Patterns in a Binary Image.

Figure 8.6 shows three temporal sequences of patterns sensed by the same bottom-level node. The two top rows display frequently occurring pattern sequences and therefore the node remembers them. The top row shows a vertical line moving to the right during 4 time steps and the middle row shows a top right corner moving diagonally down to the right. The bottom row shows uncommon pattern sequences which are ignored by the node.

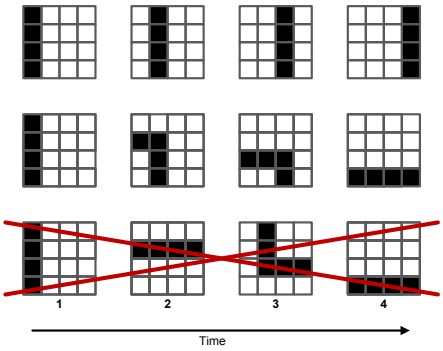

Figure 8.6 Temporal Patterns in a Binary Image.

A trained HTM model can make predictions or classifications by inferring causes from novel input.

Two important HTM concepts are *invariant representations* and *using time as a supervisor*. Both concepts are related, since using time as a supervisor aids in creating invariant representations. In order to understand the concepts, an example from computer vision will be used. Consider the task of creating a predictive model able to classify different objects in an image, for example different shapes of squares, circles and triangles. One possible solution would be to store different pictures of each shape scaled, rotated and translated in different ways. Then, during inference, the classifier could match novel pictures of shapes to its memorized shapes, based solely on a pixel-wise Euclidean distance. In practice, the model would have to store an infeasible amount of images of each shape in order to accomplish this, hence, rendering the proposed solution useless.

A better solution would be to find a way to create an internal representation of each shape using a limited amount of memory. This is exactly what the concept of invariant representations aims to accomplish using time as a supervisor in the learning process. If a model is fed a continuous, animated sequence of images (one sequence per shape) in which each shape is scaled, rotated and translated in various ways, it can learn that images sensed with little temporal difference between them must belong to the same conceptual object (shape). Compare this to the two topmost rows in Figure 8.6. Each row constitutes a sequence of four images, in which two shapes have been translated. The top row shows a vertical line translating to the right and the middle row shows a top right corner translating diagonally down to the right. By grouping similar objects together, using time as a supervisor, an invariant representation of the underlying conceptual object is obtained. The top row in Figure 8.6 is an invariant representation of a vertical line and the middle row is an invariant representation of a top right corner. This technique results

in a trained model that is immune to different scalings, rotations and translations of the underlying conceptual object during inference. In fact, once a model has been trained on continuous, animated sequences of objects, it can classify static images of the underlying object during inference (known as flash inference).

8.5 The HTM Learning Algorithms

The HTM learning algorithms [42] implement the HTM concepts described in the previous subchapter. The learning algorithms described in this subchapter are for the free version of the HTM framework (NuPIC version 1.7.1) [40]. Once again, a computer vision application will be used to exemplify the algorithms.

Figure 8.7 shows a 3-layer HTM network configured to recognize objects in a 32x32 pixel image. The input consists of a continuous, animated sequence of images. In the figure, one frame containing an 'L'-shaped object is displayed. The bottom layer (layer 1) contains 64 nodes, each mapped to a 4x4 pixel area on the image, i.e. each node has 16 inputs. As illustrated in the figure, the nodes marked with the small letters 'a' and 'b' are mapped to the 4x4 pixel areas 'A' and 'B' in the image respectively. In turn, the second layer (layer 2) nodes consist of 16 nodes, each mapped to 4 nodes in the bottom layer (layer 1). Therefore, the equivalent part of the image covered by the second layer nodes is a 8x8 pixel area, where second layer nodes 'c' and 'd' map to image areas 'C' and 'D' respectively. The equivalent image area covered by the single top layer (layer 3) node is the entire image.

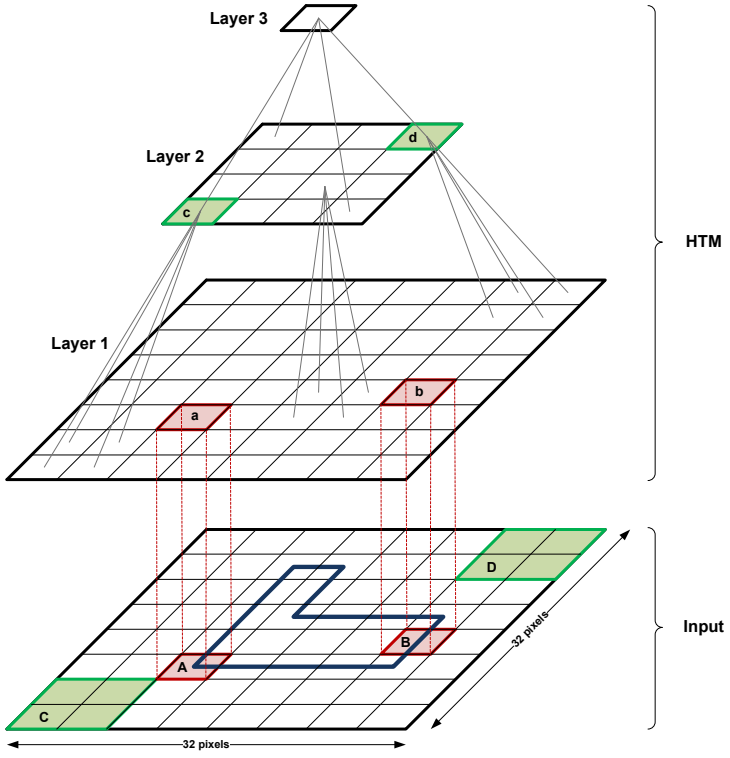

Figure 8.7 Three layer HTM network with a 32x32 pixel input image.

During training, a continuous sequence of images representing various objects is fed into the bottom layer nodes. The current pattern observed by the two bottom layer nodes marked with 'a' and 'b' in Figure 8.7 is a bottom left corner and a vertical line respectively. As the image moves, a sequence of patterns is sensed by the bottom layer nodes. For example, the node marked 'b' in the figure would sense a sequence of patterns equivalent to the topmost row in Figure 8.6 during four time frames as the blue object in Figure 8.7 moves to the right, i.e. a vertical line moving to the right.

The processed output from the bottom layer nodes becomes the input to the second layer nodes, and the processed output from the second layer nodes becomes the input to the single top level node. The input to any node, in any layer, is a temporal sequence of patterns and each node performs the same learning algorithm.

The two pooling mechanism used by a node, in order to create its invariant representations, were briefly mentioned in the previous subchapter; the *spatial pooler* and the *temporal pooler*. Each node contains a spatial pooler and a temporal pooler as shown in Figure 8.8. The input to a node reaches the spatial pooler first, followed by the temporal pooler.

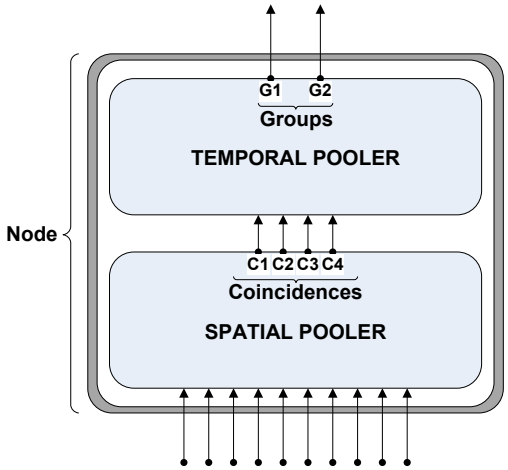

Figure 8.8 HTM node with spatial and temporal pooler.

The pooling (grouping together) of spatial patterns in the spatial pooler is based on pixel-wise similarity. It maps a potentially infinite number of input patterns to a finite number of *quantization centers*, i.e. it quantizes the spatial patterns (limits the possible number of spatial patterns stored in the spatial pooler). Compare this to clustering, where each individual input pattern is a multidimensional data point and the cluster center constitutes the quantization center. It follows that spatial pooling is also useful for removing noise from noisy inputs. The topmost patterns in Figure 8.5 are examples of quantization centers that might be stored in the bottom level nodes in the computer vision example. The quantization centers are in fact the *coincidences* mentioned in the previous subchapter. The output from a spatial pooler is in terms of its coincidences (quantization centers). The node in Figure 8.8 has learned 4 coincidences (e.g. the four topmost patterns in Figure 8.5) marked with C1, C2, C3 and C4 in the figure respectively.

The temporal pooler receives its inputs from the spatial pooler in the form of a number coincidences, and groups these patterns together based on their temporal proximity, as opposed to their pixel-wise proximity. The top row in Figure 8.6 is an example of a temporal group for a vertical line, where the individual, shifted lines have been grouped together based on their temporal proximity. Likewise, the middle row constitutes a temporal group for a right top corner, where the individual, translated versions of the corner have been grouped together based on their temporal proximity to each other. The output from a temporal pooler is in terms of its temporal groups. The node in Figure 8.8 has learned 2 groups (e.g. the two topmost pattern sequences in Figure 8.6) marked with G1 and G2 in the figure respectively. The output from a node is therefore a vector of temporal groups, which becomes the input to a higher-level node (except for the bottom level nodes which receive their input from the "outside world").

Learning in a node is a two-step process. First the spatial pooler forms its set of coincidences from its inputs. When this has been done, the temporal pooler forms temporal groups of the spatial pooler's coincidences. Figure 8.9 shows the different learning stages of a node, which are listed below.

A. Initially, a node is untrained. Its spatial pooler has not learned any coincidences (quantization centers) and its temporal pooler has not formed any temporal groups.

B. When a node starts to learn, the spatial pooler is in learning mode and the temporal pooler is inactive. At this point, the spatial pooler if fed the entire sequence of input samples. The spatial pooler then discovers frequently occurring spatial patterns in its input stream, from which it creates a number of coincidences though the quantization process.

C. Once the spatial pooler is fully trained, it switches over into inference mode and the temporal pooler becomes active in learning mode. The entire sequence of input samples is then reiterated, where the spatial pooler, now in inference mode, infers what coincidences (quantization centers) each input sample belongs to. This vector of coincidences constitutes the output from the spatial pooler and the input into the temporal pooler. The temporal pooler, now in learning mode, then starts forming temporal groups of coincidences.

D. After the temporal pooler has finished learning temporal groups of coincidences, it switches over into inference mode. The node is now fully trained with both the spatial pooler and the temporal pooler in inference mode. This entire process (A-D) is then repeated for each layer until the topmost nodes have been trained.

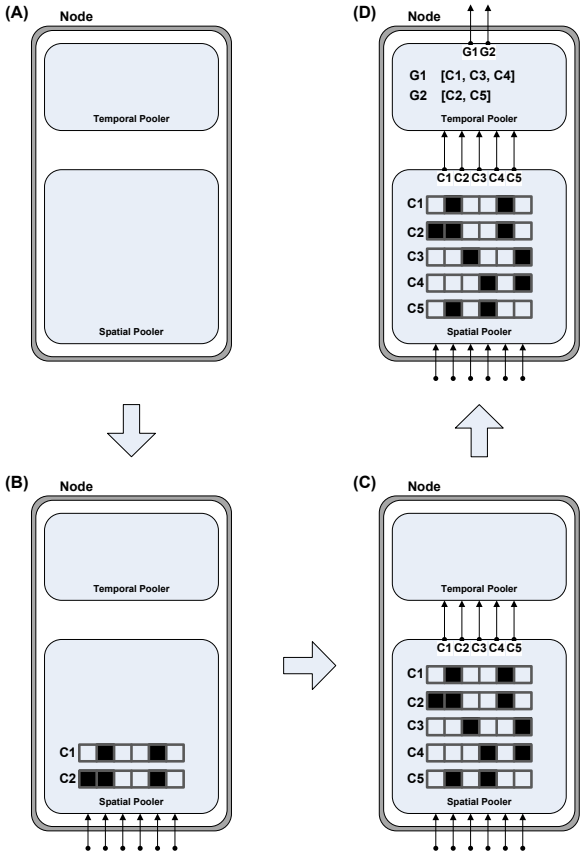

Figure 8.9 The learning stages of an HTM node.
(A) Untrained (B) Spatial Pooler in learning mode, temporal pooler inactive
(C) Spatial Pooler in inference mode and temporal pooler in learning mode
(D) Fully trained HTM node with both poolers in inference mode.

Each node has a number of configurable parameter settings. One such setting is controlled by the *maxDistance* parameter. This parameter, based on Euclidean distance, is used by the spatial pooler to determine how close (spatially) an input pattern needs to be to an existing coincidence (quantization center) for it to be considered part of that coincidence. The *maxDistance* setting in Figure 8.10 would imply that data point P_a is part of coincidence (quantization center) C_1, but data point P_b is not.

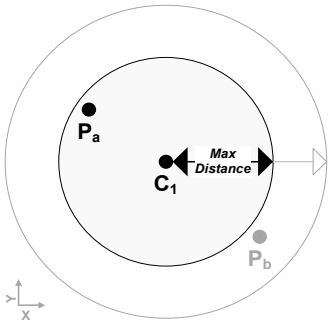

Figure 8.10 An HTM node's maxDistance setting for spatial clustering.

During learning, *maxDistance* is used to create new coincidences (quantization centers). For each input pattern, the spatial pooler compares its Euclidean distance to each existing coincidence (quantization center). If none of the existing coincidences fall within *maxDistance* of the current input pattern, a new coincidence is created. This process is repeated until the spatial pooler is fully learned or until the maximum number of coincidences have been reached. Another approach (not implemented in NuPIC 1.7.1) creates the maximum number of coincidences all at once, all randomly initialized, and uses an online k-means clustering algorithm to adjust the coincidences in the direction of each input pattern.

During inference, *maxDistance* is used to compare each input pattern to each of the learned coincidences (quantization centers). The closer the input pattern is to a coincidence, the higher the probability that it belongs to that coincidence. The output from the spatial pooler is therefore a vector of probabilities $\{c_1, c_2, ..., c_N\}$, one for each of the learned coincidences. For ease of understanding, the examples used above have all been based on noiseless inputs, i.e. each output vector has a probability of 1 in one of the elements and 0 in all others. Although, this is not the case in real-world applications.

Another configurable parameter setting for the spatial pooler is *Gaussian inference mode*, which only affects the behavior of a node during inference. If the probability that a patterns belongs to a coincidence (quantization center) falls of as a Gaussian function (Equation 8.1) of the Euclidean distance (i.e. a radial basis function), then the probability that a pattern belongs to a specific coincidence can be approximated with (Equation 8.2), where d_i is the Euclidean distance between the input pattern and the *i*th coincidence (quantization center). Sigma σ, is the standard deviation for the Gaussian curve and can be adjusted according to the amount of noise in the input stream. *Sigma* is also a configurable parameter setting of the spatial pooler and is only used in conjunction with Gaussian inference mode.

$$f(x) = \frac{1}{\delta\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{8.1}$$

$$f(x) = e^{-\frac{(x-c_i)^2}{2\sigma^2}} = e^{-\frac{d_i^2}{2\sigma^2}} \tag{8.2}$$

The temporal pooler also has a number of configurable parameter settings, including the maximum number of temporal groups it can learn. During learning, the temporal pooler learns temporal transitions of coincidences (quantization centres) by using a time-adjacency matrix (TAM). The TAM is a two-dimensional table, in which the rows and columns correspond to the coincidences occurring at time *t-1* and *t* respectively. Initially, all cells in the TAM are set to zero. For each input vector received from the spatial pooler, the temporal pooler calculates *c(t)* and *c(t-1)* using Equation 8.3, where *c(t)* is the index of the coincidence with the highest probability at time *t*, and *y(t)* is the input vector of probabilities at time *t*. Likewise, *c(t-1)* is the index of the coincidence with the highest probability at time *t-1*. The two indexes are then used to locate the corresponding cell in the TAM, using Equation 8.4, and the cell's value is incremented by one. In other words, if coincidence *c(t)* follows coincidence *c(t-1)* in time, its corresponding value in the TAM is incremented by one.

Periodically, the TAM is normalized along its rows to produce a true first-order transition matrix (i.e. each cell contains a probability value between 0.0 and 1.0).

$$c(t) = argmax\, y(t) \tag{8.3}$$

$$TAM(c(t-1), c(t)) \tag{8.4}$$

Figure 8.11 depicts a time-adjacency matrix (TAM) where 4x4 pixel patterns have been displayed along the rows and columns for illustrative purposes. Bright squares in the TAM represent cells with high counts (high probabilities) whereas dark squares represent cells with low counts (low probabilities). Consider the sixth row down from the top of the TAM. The row contains two bright squares in columns 4 and 5. With support of Equation 8.4, showing the indices into the TAM, the vertical line (displayed to the left of the row in the figure) is highly likely to be followed by either a shift to the left (the pattern displayed above the fourth column) or highly likely to remain stationary (the pattern displayed above the fifth column). Similarly, by inspecting rows 2, 3 and 4, it is obvious that a bottom right corner is most likely to move to the left.

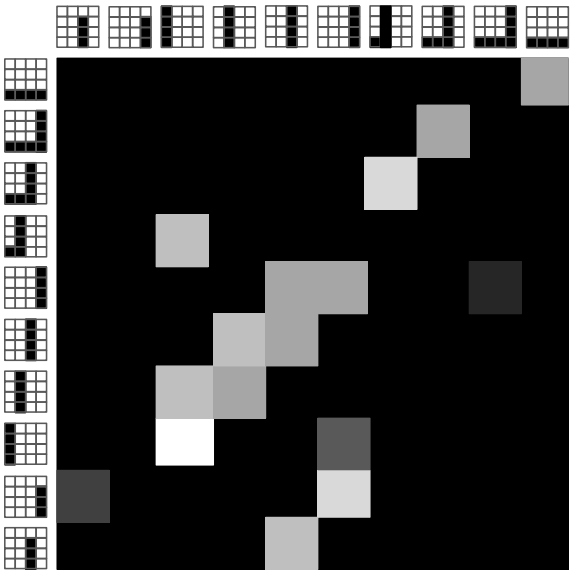

Figure 8.11 Time-adjacency matrix.

At the end of the learning phase, the temporal pooler forms temporally coherent groups by partitioning the time-adjacency matrix (TAM). Coincidences in the same group are highly likely to follow each other in time. If $\{D_1, D_2, ..., D_N\}$ is a disjoint set of coincidences, Equation 8.5 can be used to measure how good a group is, where t_i is the time-adjacency measure, n_i is the number of elements in partition D_i, and $T(k,m)$ is the (k,m)th entry in the TAM. From the per-partition similarity measure, a global objective function (Equation 8.6) can be defined (bigger values of t_i and J are better).

$$t_i = \frac{1}{n_i^2} \sum_{k \in D_i} \sum_{m \in D_i} T(k, m) \tag{8.5}$$

$$J = \frac{1}{2} \sum_{i=1}^{N} n_i t_i \tag{8.6}$$

Since enumerating all possible combinations of groupings is infeasible, a greedy algorithm is used to separate each coincidence into a group. The greedy algorithm is based on directed graphs, where each coincidence (quantization point) is considered to be a node in a connected graph):

1. Find the most connected quantization point that is not yet part of a group (whose corresponding row in the time-adjacency matrix has the greatest sum).
2. Find the *topNeighbors* (*topNeighbors* is a configurable parameter) most connected quantization points (that have the greatest row sums in the matrix), connected to the chosen point in step 1.
3. Repeat step 2 for each of the newly-added quantization points, or until *maxGroupSize* is reached (*maxGroupSize* is a configurable parameter).
4. Add the quantization points to a new temporal group.
5. Repeat steps 1-4 until all quantization points have been grouped.

The greedy algorithm is illustrated in Figure 8.12 and explained in the text below. Thicker lines in the graphs symbolize nodes with stronger connections (have greater sums in the time-adjacency matrix).

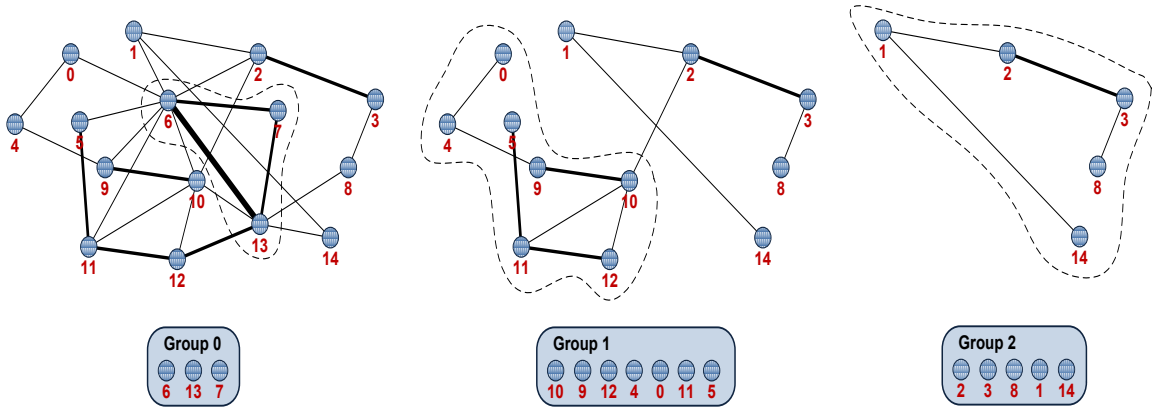

Figure 8.12 Greedy algorithm for forming temporal groups (topNeighbors = 2).

With topNeighbors set to $N_{top}=2$, the first temporal group (Group 0) is formed as follows:

1. Add the most connected point to Group 0. Looking at the leftmost graph, point 6 is most connected with 9 connections to other nodes. Therefore point 6 is added to Group 0.
2. Add the N_{top} most connected (strongest points, i.e. with thicker lines) connected to point 6. Point 13 followed by point 7 are therefore added to Group 0.
3. Add the N_{top} most connected (strongest points, i.e. with thicker lines) connected to point 13. These two points are 6 and 7 which have already been added to Group 0.
4. Add the N_{top} most connected (strongest points, i.e. with thicker lines) connected to point 7. These two points are 6 and 13 which have already been added to Group 0.
5. Since no more points can be added, Group 0 is complete.
6. Remove the points in Group 0 from the graph, which results in the middle graph in Figure 8.12.

Similarly, the next group (Group 1) is created using the middle graph in Figure 8.12:

1. Add the most connected point (point 10) to Group 1.
2. Add the N_{top} most connected points for point 10 (point 9 and 12) to Group 1.
3. Add the N_{top} most connected points for point 9 (point 10 and 4) to Group 1 (point 10 already added to Group 1).
4. Add the N_{top} most connected points for point 4 (point 9 and 0) to Group 1 (point 9 already added to Group 1).
5. Add the N_{top} most connected points for point 0 (point 4 which has already been added to Group 1).
6. Add the N_{top} most connected points for point 12 (point 10 and 11) to Group 1 (point 10 already added to Group 1).
7. Add the N_{top} most connected points for point 11 (point 5 and 12) to Group 1 (point 12 already added to Group 1).
8. Since no more points can be added, Group 1 is complete.
9. Remove the points in Group 1 from the graph, which results in the rightmost graph in Figure 8.12.

Finally, Group 2 is created from the remaining points (rightmost graph in Figure 8.12):

1. Add the most connected point (point 2) to Group 2.
2. Add the N_{top} most connected points for point 2 (point 3 and 1) to Group 2.
3. Add the N_{top} most connected points for point 3 (point 2 and 8) to Group 1 (point 2 already added to Group 2).
4. Add the N_{top} most connected points for point 1 (point 2 and 14) to Group 1 (point 2 already added to Group 2).
5. Since no more points can be added, Group 2 is complete.
6. Remove the points from the graph, which gives an empty graph, hence, the time-adjacency matrix has been fully partitioned into 3 disjoint temporal groups.

During inference, the output from the temporal pooler is a probability distribution over the temporal groups, i.e. a vector containing as many elements as there are groups. Each element contains the probability value associated with the coincidence having the highest probability value within that group.

Figure 8.13 shows the inference procedure for a node receiving three consecutive patterns. In this case, there is no noise in the input, which means that the coincidences can be matched exactly. The first (left) input pattern is recognized as belonging to coincidence *c4*. Therefore, the output vector contains a probability of 1 in vector element 4 and zero probabilities for all other elements. Similarly, the temporal pooler recognizes its input as belonging to temporal group *g2*, so its output has a probability of 1 for element 2 and zero probability for all other elements. The other two input patterns (middle and right) follow the same scheme. Even though the node senses three different patterns of a bottom-left corner (the output from the spatial pooler is different for all three patterns), all patterns belong to the same temporal group. Therefore, the output from the node is the same during all three time frames. This is the essence of learning invariant representations of the same cause.

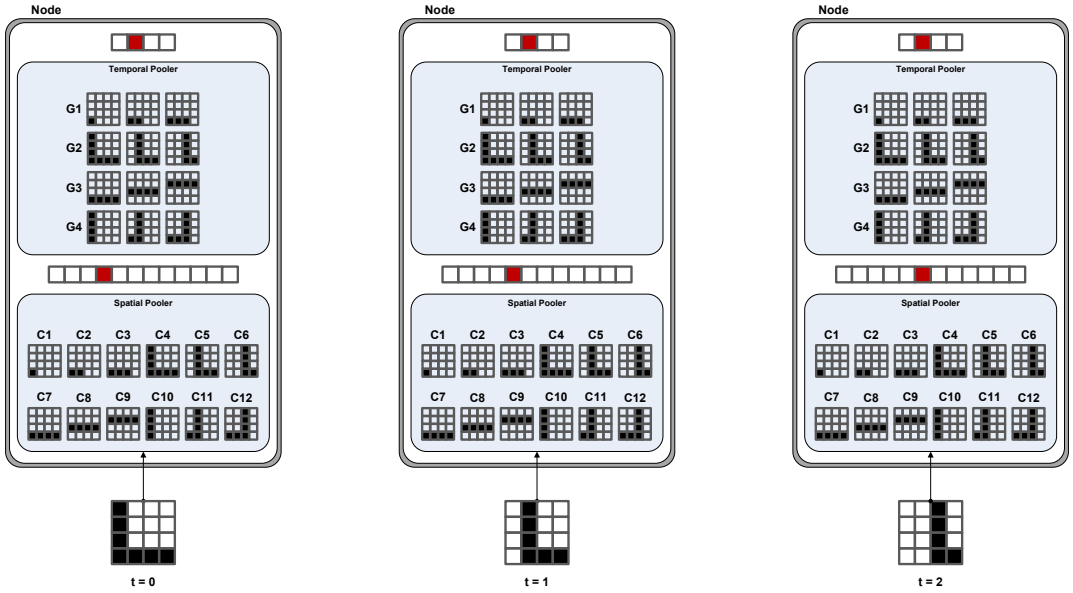

Figure 8.13 HTM node's inference procedure for three consecutive patterns.

In Figure 8.14, the node used in Figure 8.13 has been duplicated. Both nodes have then been connected to one node further up the hierarchy to create a two-layer HTM network. As in Figure 8.13 both bottom-layer nodes receive three consecutive, but different, patterns. The left, bottom layer node receives the same pattern as in Figure 8.13, whereas the pattern for the right node has been rotated horizontally. Once again, there is no noise in the inputs in this case, which means that the coincidences can be matched exactly. The first (left) input pattern is recognized as belonging to coincidence *c4* for the left node and as *c10* for the right node. Therefore, the output vector contains a probability of 1 in vector element 4 for the left node and in element 10 for the right node. All other elements have zero probabilities. The temporal poolers recognize their inputs as belonging to temporal

group *g2* for the left node and as *g4* for the right node, their outputs have a probability of 1 for element 2 in the left node and for element 4 in the right node. All other elements have zero probabilities. The other two input patterns (middle and right) follow the same scheme. The outputs from both bottom layer nodes are concatenated and fed as the input to the higher layer node. Since the two four-element outputs have been concatenated, the input to the higher layer node contains eight elements, with a value of 1 in element number 2 (from the left node) and element number 8 (from the right node). All other elements have zero values. The higher layer node, which is in learning mode, realizes that its current input during time step 0 is a new pattern, and adds it to its internal memory as a spatial pattern. During time steps 1 and 2, the same input pattern is received and recognized, therefore no additional patterns are stored. The same learning/inference procedure is performed by all nodes further up the hierarchy until the topmost level has been reached. Hence, invariant representations are learned by each node in the hierarchy.

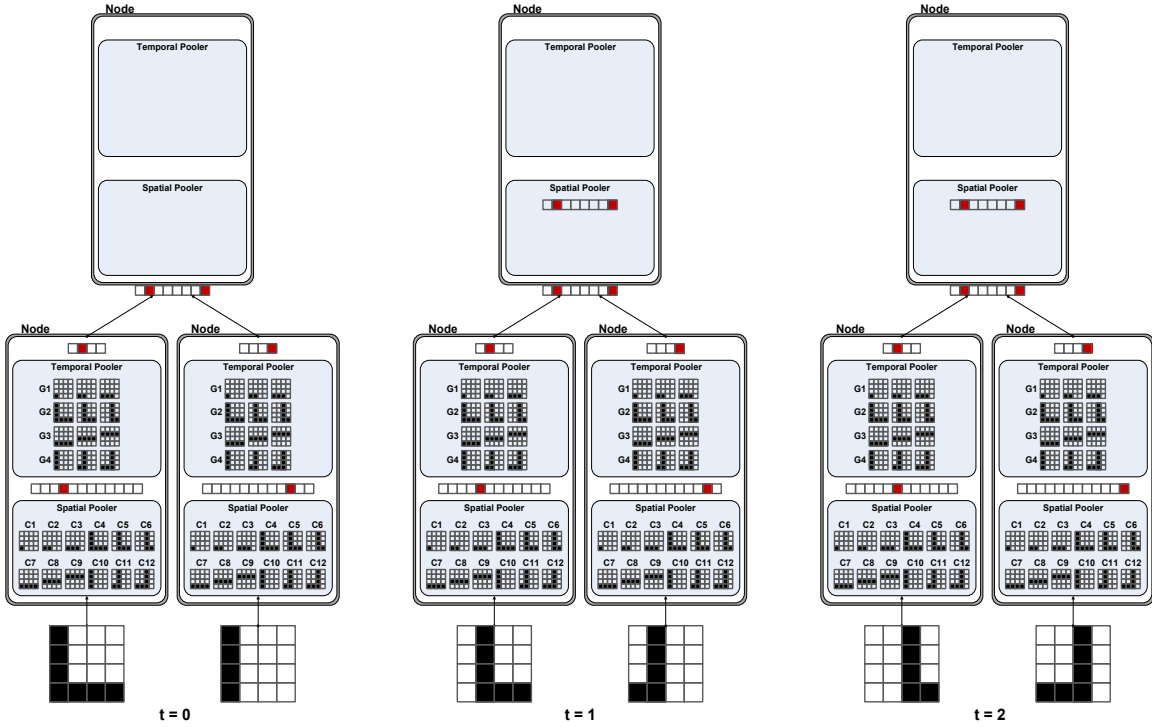

Figure 8.14 Inference/learning procedure for nodes in a hierarchy.

When supervised learning is used to train an HTM network, a supervisory signal (vector of target classes) is used to form temporal groups in the topmost node. Usually a special node implementing the Naïve Bayes algorithm is used as a top level node for classification purposes. For regression, a linear output is used instead (just as in the MLP network). Figure 8.15 shows a typical configuration of an HTM network for a classification problem. A sensory node (1) receives the input vectors and distributes them evenly over the bottom level nodes (2-5). These nodes process their inputs and send their outputs to the level 2 nodes (6 and 7) which, in turn, process their inputs and send their outputs to the top-level node (8). The top-level node also receives the class vector (category vector) from the category input node (10) to enable supervised learning (this is

of course omitted for the test set). The top-level node then outputs one classification for each input. An effector node (9) can be used to receive the output from the top-level node and make some decision based on the value of its inputs. An effector node could simply write the contents of its input to a file on disk or send a buy or sell order to an electronic exchange in an automated trading system.

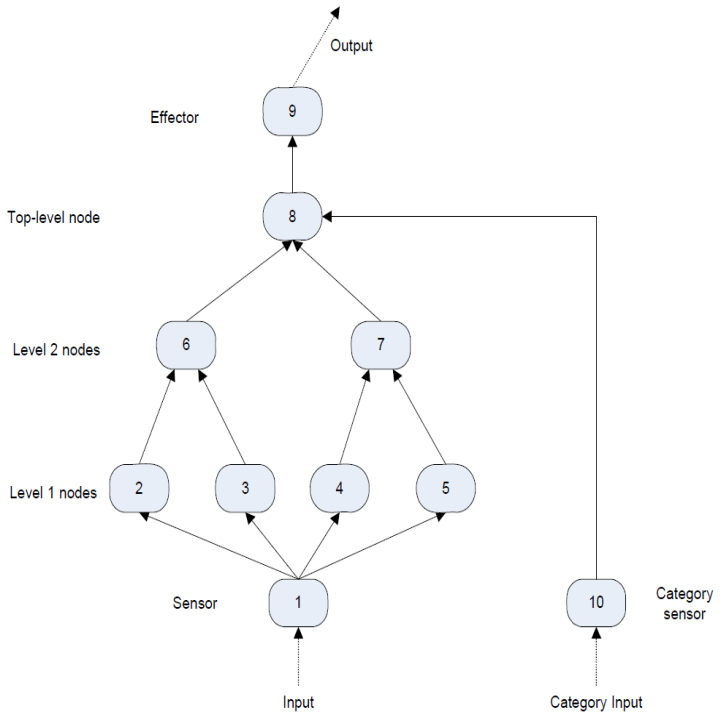

Figure 8.15 HTM network configured as a classifier.

Both neural networks and HTMs contain a number of parameters than can be tweaked in order to optimize performance. If domain knowledge is available for a specific application area and the designer of the system has a lot of experience working with neural networks or HTMs, then a manual optimization procedure might be preferred. Otherwise, an evolutionary approach, using genetic algorithms, can be used to tune the parameters of the network. Evolutionary learning is explained in the next chapter.

9 Evolutionary Learning

In 1859 Charles Darwin published a book describing his evolutionary theory of natural selection and the survival of the fittest [43]. During the nineteenth century, this radical new idea of the origin and evolution of spices met a lot of resistance, especially from theologians. After all, during that time period, who could believe that the introduction of new species was a natural process rather than a miraculous one? Fast-forwarding one and a half centuries, we now know that evolution is a natural part of biological systems. Furthermore, the ideas of biological evolution have found their way into technology in the form genetic algorithms. This chapter follows the same layout as the two previous chapters, starting off by introducing the biological system, more specifically genetics, followed by the intricacies of the genetic algorithm.

9.1 The Biological System

All content in this subchapter is adopted from [23].

A human cell is roughly comprised of a *cell membrane* which defines the outer boundary of the cell, a gel-like substance called the *cytoplasm* that covers the inside of the cell and is supported by the *cytoskeleton*, and a cell *nucleus* which constitutes the core of the cell. The cytoplasm contains microscopic structures called *organelles* which are the cell's "organs". Besides the organelles, another important, pervasive substances in the cytoplasm is *proteins*, which are the building blocks of all life.

Inside the nucleus, 23 *chromosome* pairs are housed, where each pair is made up of two *DNA* (*deoxyribonucleic acid*) strands that twist around each other to create a spiral-shaped structure called a *double helix*. Specific portions of a DNA sequence are called *genes*, which contain the genetic information for making proteins.

Inherited traits for an individual are passed from parents to offspring through the genetic information stored in the form of DNA molecules in the parents' chromosomes. This genetic information is also called the *genetic code*, which instructs cells how to synthesize enzymes and protein molecules. The process of decoding a gene in order to synthesize a certain protein is called *gene expression*. Two sex cells, one from each parent, fuse with each other to create the first cell of an offspring. During the fusion process, one chromosome chromatide from each parent are combined with each other, sharing genetic information from each parent. This mechanism is commonly known as a *crossover* operation. Through a process known as *mitosis* (cell division), the DNA in the nucleus undergoes *replication*, in which the DNA is copied to new cells as the individual develops, hence preserving the genetic information in each and every cell. Collectively, all the DNA within a cell is called the *genome*.

If the DNA double helix is straightened out, its shape can be compared to a ladder. The two sides of the ladder, comprised of alternating groups of sugar and phosphate portions, constitute the backbone part of the DNA molecule. The steps of the ladder symbolize a pair of *nucleotides*, one in each DNA strand, that bind together to maintain the structural shape of the DNA molecule. In other words, the nucleotides run along the sides of the ladder (DNA backbone) where each step is formed by binding two nucleotides, one in each end of the step, together. Four different types of nucleotides exist in each DNA strand; *adenine* (A), *thymine* (T), *cytosine* (C) and *guanine* (G). An adenine will only bond to a thymine, and a cytosine will only bond to a guanine. Therefore, the sequence of nucleotides along one strand bond with a complimentary sequence of nucleotides on the other strand. A sequence of nucleotides along a DNA strand specifies a specific protein's amino acid sequence (proteins are made up of sequences of amino acid groups, just like DNA strands are made up of sequences of nucleotide base groups).

During the DNA replication process (during mitosis), enzymes break the bond between the nucleotides pairs (break the steps in the ladder) so that the double helix unwinds. Another enzyme, called *DNA polymerase*, then brings in new DNA nucleotides forming complimentary pairs with the exposed nucleotide bases. A third enzyme pieces together

the two missing backbones (sides of the ladder) and the result is two identical DNA molecules. Sometimes an error occurs in the replication process, where sequences of nucleotides along a DNA strand contain faulty nucleotide groups. This is called a *mutation*, and is normally detected and repaired by error-checking enzymes, although sometimes this check fails.

Gene expression, decoding the sequence of nucleotides in a gene followed by encoding sequences of amino acids in proteins (protein synthesis), is what manifests the specific traits for an individual. Twenty different types of amino acids can be represented by specific sequences of three nucleotides in the DNA. Other sequences of nucleotides determine the beginning and end of the synthesis of a specific protein. The gene expression process consists of the two steps, *transcription* and *translation*.

During the heredity DNA replication process, gene copies can be identical or slightly different in DNA sequence. Two genes that differ slightly in sequence are called *alleles*. The combination of alleles for one gene, or many, is called an individual's genotype. The traits associated with a specific genotype is called the individual's phenotype.

9.2 The Genetic Algorithm

In 1975, John Holland was inspired by Darwin's work and contemporary knowledge obtained in the field of genetics. He decided that ideas from biological evolution could be applied to the technical field and created the genetic algorithm [44]. John Koza later expanded on Holland's algorithm in 1992 to create the closely related field of genetic programming [45]. This chapter only describes Holland's genetic algorithm.

The basic genetic algorithm works on a population of individuals that are evolved over a number of generations. The fittest individuals in each generation live longer, are more attractive and have a higher chance of mating with other fit individuals in order to produce offspring. This is in direct analogy with Darwin's evolutionary theory of natural selection and the survival of the fittest.

In the simplified genetic model, each individual has one chromosome (single strand) that contains a number of genes. When two fit parents are *selected* for *reproduction*, they swap a few genes in their chromosomes with each other (*crossover*) which becomes the building blocks of the resulting chromosome in their offspring. One version of each gene (*allele*) from both parents is hence kept in the offspring's chromosome. Just as in the biological system, there is always a slight chance that a gene is *mutated*. The concepts of selection, reproduction, crossover and mutation have been brought over from the biological system to enable exploration and exploitation in the technical system. The genetic algorithm performs a search in the solution space by altering the genome in order to find the fittest individual.

The first task, when using the genetic algorithm, is to represent the problem that needs to be solved in the form of a chromosome. Holland argued that the best way to do this is in the form of a bit string. Each bit in the string constitutes a gene, and its value (allele) is chosen from an alphabet of a given size. Holland used a binary alphabet, in which a gene

can take on the value of 0 or 1, although contemporary solutions use alphabets consisting of the entire range of real numbers. Following in Holland's footsteps, a binary alphabet will be used in the following text. Therefore, the problem to a solution needs to be encoded as a binary string of a certain length. As an example, borrowed directly from [45], the problem of choosing the right combination of the price of a burger meal, the type of drink served and the type of service provided at four different burger restaurant locations can be encoded as a bit string of length 3. Each allele for the three genes has two values; price (high or low), drink (cola or wine) and speed of service (fast or leisurely). The problem, represented as a chromosome, is depicted in Figure 9.1.

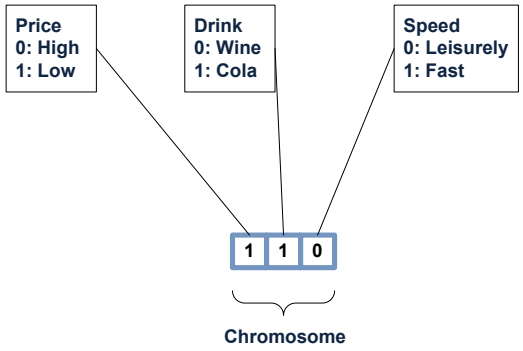

Figure 9.1 Chromosome representation of the restaurant problem.

Once the problem has been represented as a bit string, an evaluation function needs to be defined that answers the question: "how fit is an individual?". This fitness function is the only direct link between the general intricacies of the genetic algorithm and the real world. It needs to provide a fitness value for each individual in the population based on the value of its genes, i.e. how fit is an individual for solving the problem? Therefore, a lot of thought needs to go into the definition of the fitness function. In the case of the burger restaurant, the fitness for an individual could be defined as the profit made by using the information stored in an individual's genes to run the business during a certain time period. Or, using a simpler example, if the fitness function was defined as the integer equivalent to the binary number stored within the chromosome displayed in Figure 9.1, the fitness for the chromosome would be $f = 1*4 + 1*2 + 0*1 = 6$.

Now that the problem has been represented in chromosomes and a fitness function has been defined to measure the fitness of an individual, a population of a certain size is determined. Once this has been done, each individual in the population is randomly initialized and inserted into the first generation. The fitness of each individual is then measured and the fittest are bread with each other in order to produce offspring for the next generation, as illustrated in Figure 9.2. This process of breeding new generations and measuring their fitness is repeated for a certain amount of generations or until a solution is found (if the correct solution is known). During each generation, the total fitness of the population increases by applying the basic genetic operators; selection, reproduction, crossover and mutation.

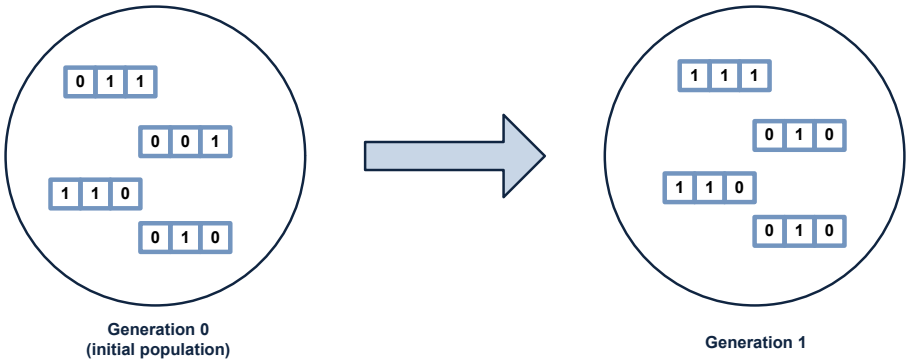

Figure 9.2 Two generations of the chromosome population.

The fittest individuals in the current generation are selected for reproduction and moved over to a mating pool where they can breed with each other to create the next generation of offspring. Different selection methods exist, where *fitness proportional selection* was advocated by Holland. Using fitness proportional selection, chromosomes are selected probabilistically, where the probability of an individual being selected is proportional to its fitness. The fitness proportional selection function is shown in Equation 9.1, where the probability p^n of selecting an individual n is calculated as the fitness F^n for individual n divided by the sum of the total fitness for the entire population.

$$p^n = \frac{F^n}{\sum_N F^N} \tag{9.1}$$

If the fitness function can take on positive and negative values, Equation 9.1 will not work. In such cases, Boltzmann selection can be used instead as shown in Equation 9.2, where s is an adjustable strength parameter. The equation is essentially the same as the softmax function.

$$p^n = \frac{e^{(sF^n)}}{\sum_N e^{(sF^N)}} \tag{9.2}$$

Once the fittest individuals have been selected into the mating pool, they are paired up with each other for reproduction (or recombination). This is done using the crossover operator. Crossover happens with a certain probability, usually somewhere around 0.8, and comes in different flavors. The most common type of crossover is *single point crossover*. In single point crossover, one position in the bit string is randomly selected. Each parent is divided in two at the crossover point and the latter part in each parent is swapped with the other. This creates two offspring each comprised of different parts of their parents. This mechanism performs global exploration in the solution space, in hope of finding a fitter individual. Another variant of crossover is *multipoint crossover*. In multipoint crossover, two positions in the bit string are randomly selected, where the middle part (between the two crossover points) of the bit string in each parent is swapped with the other. The two common crossover operations are shown in Figure 9.3.

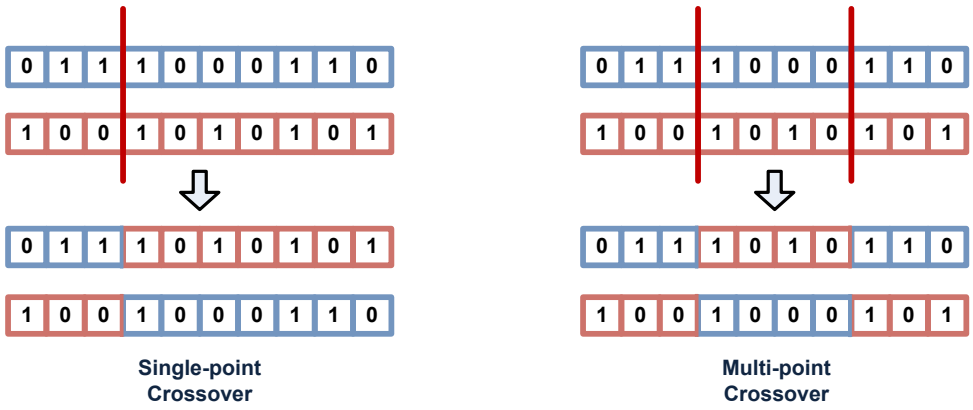

Figure 9.3 Crossover operations.

Following the crossover operation, there is a relatively small probability that one gene in each offspring is mutated. A rule of thumb is to chose the probability as the reciprocal of the length of the bit string (number of genes), i.e. if the length of the bit string is L, the mutation probability is $1/L$. For a binary bit string, the mutation operator is implemented as a bit-flip, i.e. if the selected element's value is a 1 it is exchanged for a 0, and if the value is 0 it is exchanged for a 1, as depicted in Figure 9.4. The mutation operator constitutes an exploitation of the bit strings, which performs a local random search in the solution space. If the alphabet of the bit string consists of real numbers, a commonly used mutation operation is to randomly sample from a standard normal distribution (Gaussian) and then add the result to the mutated element.

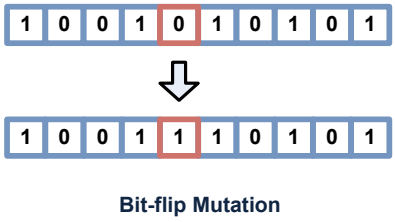

Figure 9.4 Mutation operator.

One problem with the current setup is that the fittest individuals can be destroyed during crossover. This can cause the total fitness of the population to decrease between two generations. In order to rectify this problem, *elitism* or *tournaments* can be employed. When elitism is used, a certain number of the fittest individuals are copied into the next generation without any crossover operation. Tournaments let parents and offspring fight it out to see which individuals are the fittest. If a parent beats an offspring, the parent is copied into the next generation instead. Elitism and tournaments therefore preserve the fittest individuals through the generations. The downside with these approaches is that it hampers exploration, in effect making it hard to escape from a local maximum. Therefore, there is a tradeoff between using elitism and tournaments or not.

To summarize what has been said above, the genetic algorithm is defined as follows:

1. Generate an initial population of randomly initialized individuals of size P, each individual having length L of a certain alphabet. This is the population for generation 0.
2. Evaluate the fitness of each individual in the first generation using the fitness function.
3. Use the selection function to select two individuals from the current generation.
4. Recombine the selected individuals to produce two offspring.
5. Mutate the offspring.
6. Add the offspring to the new generation, or use elitism/tournaments.
7. Repeat steps 3-6 until the size of the new generation is P.
8. Evaluate the fitness of each individual in the current generation using the fitness function.
9. If the stopping criteria has not been met, go to step 3.
10. Stop the genetic algorithm and return the fittest individual as the solution.

The genetic algorithm can be used to optimize the parameters of a classification or regression model. This will be used to optimize the neural network and HTM model parameters in chapter 12. Before that, the next chapter introduces related research work that has been carried out regarding predictive modeling of financial markets.

10 Related Work

When it comes to neural networks and the genetic algorithm, there are plenty of studies available with regards to predictive modeling of financial markets, since these machine learning technologies have been around for a relatively long time. Hierarchical Temporal Memory, on the other hand, is a relatively new machine learning technology and few such studies are available. Therefore, the related work for the HTM technology includes studies from various application areas.

10.1 Neural Networks and Evolutionary Learning

In [46], three feature subset selection methods were investigated in producing the inputs to backpropagation neural network models for stock prediction; Principal Component Analysis (PCA), Genetic Algorithms (GA) and decision trees (CART). The study showed the importance of this preprocessing step in enhancing the performance of predictive neural network models, where 80% of the original 85 features were filtered out using a combination of all three feature selection techniques. Two other studies, [47] and [48], used information gain to select the most predictive features for training neural network models for predicting the stock market. Independent component analysis (ICA) was used in [49] to remove noisy components from the inputs fed to neural networks for predicting the direction of the stock market.

A sample study using neural networks together with the backpropagation algorithm for forecasting stock indices can be found in [50]. In [51] neural networks are used to predict the Brazilian stock market. One paper combines neural networks with technical analysis using a volume frame of reference to predict stock prices [52], and in [53] a hybrid model is developed for stock market prediction by combining neural networks and genetic

algorithms. General recommendations for optimizing neural networks using the genetic algorithm can be found in [54], [55].

Cross validation techniques and classifier ensamples have also been used in financial forecasting. In [56], the use of cross validation for time series predictor evaluation is investigated, and in [57] classifier ensembles of neural networks, decision trees and logistic regression are used to predict stock returns. In [58] various approaches of forecasting with artificial neural networks is investigated.

An ambitious survey of 100 published articles using neural networks for stock market prediction is presented in [59]. The survey compares different approaches based on; input data, forecasting methodology, performance evaluation and performance measures used.

On a historical note, one paper gives an overview of the development of neural networks from its introduction to the 1990s [60].

10.2 Hierarchical Temporal Memory

Following the publication of Hawkins' memory-prediction theory of brain function [33], supported by Mountcastle's unit model of computational processes across the neocortex [32], George and Hawkins implemented an initial mathematical model of the Hierarchical Temporal Memory (HTM) concept and applied it to a simple pattern recognition problem as a successful proof of concept [36]. This partial HTM implementation was based on a hierarchical Bayesian network, modeling invariant pattern recognition behavior in the visual cortex.

Based on this previous work, an independent implementation of George's and Hawkins' hierarchical Bayesian network was created in [61], in which its performance was compared to a backpropagation neural network applied to a character recognition problem. The results showed that a simple implementation of Hawkins' model yielded higher pattern recognition rates than a standard neural network implementation, hence confirming the potential for the HTM concept.

The HTM model was used in three other character recognition experiments [62–64]. In [63], the HTM technology was benchmarked against a support vector machine (SVM). The most interesting part of this study was the fact that the HTM's performance was similar to that of the SVM, even though the rigorous preprocessing of the raw data was omitted for the HTM. A method for optimizing the HTM parameter set for single-layer HTMs with regards to hand-written digit recognition was derived in [64], and in [62] a novel study was conducted, in which the HTM technology was employed to a spoken digit recognition problem with promising results.

The HTM model was modified in [65] to create a Hierarchical Sequential Memory for Music (HSMM). This study is interesting since it investigates a novel application for HTM-based technologies, in which the order and duration of temporal sequences of patterns are a fundamental part of the domain. A similar approach was adopted in [66],

[67] where the topmost node in a HTM network was modified to store ordered sequences of temporal patterns for sign language recognition.

Few studies have been carried out with regards to the HTM technology applied to algorithmic trading. Although, in 2011 Fredrik Åslin conducted an initial evaluation [68], in which he developed a simple method for training HTM models for predictive modeling of financial time series. His experiments yielded a handful of HTM models that showed promising results. The work done in [1] expanded on Åslin's initial study, in which it was shown that the HTM-based algorithm was profitable across bullish-, bearish and horizontal market trends, yielding results at least as good as its benchmark (neural networks). Although, the previous work lacked any attempt to produce near optimal trading models, which is the main topic of this paper.

PART III
Experimental Design and Execution

11 Hypotheses

This chapter elaborates on a number of hypotheses with support of the theory presented in the previous chapters. The resulting hypotheses are defined using conditional propositional logic and are associated with the research objectives. This enables an easy assessment of the results, obtained from the experiments, and determines if the research objectives have been fulfilled.

First, some terminology needs to be introduced. In the experiments, introduced in chapters 12 and 13, four validation datasets and four test datasets are used over three iterations. Propositional variables will be denoted as; a *v* for a validation dataset from the set *V* (11.1), a *t* for a test dataset from the set *T* (11.2) and an *i* for an iteration from the set *I* (11.3). For each dataset, the genetic algorithm evolves a population of many HTM models, *M* (1.4) and artificial neural network (ANN) models, N (1.5) from which the best model, *m* and *n* is chosen. A model is associated with its performance score, *PNL*, which is its profit-and-loss on a specific dataset. An iteration averages the *PNL* for a model over all four datasets.

$$v \in V, V = \{1,2,3,4\} \tag{11.1}$$

$$t \in T, T = \{1,2,3,4\} \tag{11.2}$$

$$i \in I, I = \{1,2,3\} \tag{11.3}$$

$$m \in M, M = \{any\ HTM\ model\} \tag{11.4}$$

$$n \in N, N = \{any\ ANN\ model\} \tag{11.5}$$

A quantified conditional statement in propositional logic is expressed according to (11.3), where *n* is a propositional variable, P_1 and C_1 are propositional functions taking one propositional variable, *n*. P_1 is called the premise and C_1 is called the conclusion. The right arrow, → denotes a conditional statement, stating that if the premise $P_1(n)$ is true, then the conclusion $C_1(n)$ is implied. The existential quantifier, ∃ states that, for some *n*, the conditional statement is true. If the universal quantifier, ∀ is used instead, it states that for all *n*, the conditional statement is true. The variable *n* is associated with a universe of discourse, i.e. if the universe of discourse is defined as *n=[1,2,3,4,5]*, *n* can take on the integer values ranging from 1 to 5. With this universe of discourse, the conditional propositional statement in (11.3) reads: "*For some n, if premise $P_1(n)$ is true, then this implies conclusion $C_1(n)$*". This syntax will be used to give mathematically correct expressions of the hypotheses in this chapter.

$$\exists n\ (P_1(n) \rightarrow C_1(n)) \tag{11.3}$$

11.1 Hypothesis 1

Returning to the theory, chapter 3, regarding the nature of financial markets provided the theoretical foundation necessary to understand financial market macro- and microstructure. The Efficient Market Hypothesis (EMH) was also introduced, stating that financial markets are efficient to such a degree that it is impossible to consistently

outperform the market. Furthermore, the EMH claims that there is no autocorrelation in financial time series, i.e. that there is no serial dependencies between prices, therefore the direction of financial markets cannot be predicted by projecting historical price patterns into the future. It was also mentioned that the trading community has refuted the EMH through empirical results, hence proving that real financial markets are not as efficient as suggested by the EMH.

As a consequence of the trading community refuting the EMH, a number of various methods for analyzing financial markets have been developed. These are explained in chapter 4. In the theory, technical analysis, which has its roots in Dow Theory, is expressed as a suitable method for analyzing financial markets from a technical perspective. Technical analysis assumes that prices in financial time series are correlated, hence price patterns in historical financial data can be used to predict future market movements. This suggest that financial forecasting can be applied to any asset class (asset classes are detailed in chapter 5).

From the theory in chapter 8, it is suggested that the Hierarchical Temporal Memory model can be used to discover and learn invariant representations of patterns in time series using time as a supervisor. This can be done using supervised learning following the data mining process according to chapter 6.

This theoretical background raises the question if it is possible to create predictive Hierarchical Temporal Memory (HTM) models capable of exploiting market inefficiencies in order to yield a profitable return. This constitutes the first hypothesis (H.1):

Let P_1 denote the statement "*HTM model m yields a positive average return on dataset d*" and C_1 denote the statement "*HTM model m is capable of exploiting financial market inefficiencies*". This gives the conditional statement; if the premise, P_1 is true, it necessarily implies the conclusion, C_1. The universe of discourse is the finite set v={1,2,3,4} of validation sets (11.1) in this thesis, and any HTM-based model, *m* from the set *M* (11.4). Hypothesis (H.1a) is thus defined as:

"*For some HTM model and some validation set, if the HTM model yields a positive average return on the dataset, then the HTM model is capable of exploiting financial market inefficiencies*".

$$\exists m, v \left(P_{1a}(m, v) \rightarrow C_{1a}(m)\right) \qquad m \in M,\ v \in V \tag{H.1a}$$

where,
P_{1a} = *"HTM model m yields a positive average return on validation dataset v*".
C_{1a} = *"HTM model m is capable of exploiting financial market inefficiencies*".

Although, this is not enough to verify that HTM models are capable of exploiting financial market inefficiencies *consistently*. In order to do this, multiple iterations must be carried-out over the datasets, as described in chapter 6. Therefore, in order to prove that the implication holds consistently, hypothesis (H1.b) must be made:

"*For some HTM model and some validation set over all iterations, if the HTM model yields a positive average return on the dataset, then the HTM model is capable of consistently exploiting financial market inefficiencies*".

$$\exists m, v \forall i \left(P_{1b}(m, v, i) \rightarrow C_{1b}(m)\right) \quad m \in M,\ v \in V,\ i \in I \qquad \text{(H.1b)}$$

where,
P_{1b} = *"HTM model m yields a positive average return on validation dataset v in iteration i*".
C_{1b} = *"HTM model m is capable of consistently exploiting financial market inefficiencies*".

These hypotheses are directly associated with research objective (RO.1) in chapter 1.4 and research question (RQ.1) in chapter 1.1.

11.2 Hypothesis 2

According to the HTM theory in chapter 8, HTM models have multiple parameters that can be fine-tuned in order to produce optimal models. Furthermore, chapter 9 presents an efficient evolutionary algorithm, the genetic algorithm (GA), which is able to explore and exploit a solution space consisting of a population of individuals, representing the solution to a problem. This raises the question if predictive HTM models can be optimized for financial markets using the genetic algorithm. This constitutes the second hypothesis (H.2a):

"*For some GA-optimized HTM model and all validation sets, if the HTM model yields a positive average return on the datasets, then the HTM model can be optimized for financial markets using the GA algorithm*".

$$\exists m \forall v \left(P_{2a}(m, v) \rightarrow C_{2}(m)\right) \quad m \in M,\ v \in V \qquad \text{(H.2a)}$$

where,
P_{2a} = *"GA-optimized HTM model m yields a positive average return on validation dataset v*".
C_{2} = *"HTM model m can be optimized for financial markets using the GA algorithm*".

Although, the theory clearly states, that the GA is stochastic (non-deterministic) in nature, i.e. if the same input data is fed to the GA, it will most likely not produce the exact same result twice. Therefore, a second, more rigorous, hypothesis needs to be created that states that the stochastic GA algorithm optimizes HTM models *every time* it is employed. According to the theory in chapter 6, this can be done by iterating over the optimization process multiple times. Therefore, the second hypothesis is restated as (H2.b):

"*For some GA-optimized HTM model and all validation sets in all iterations, if the HTM model yields a positive average return on the datasets, then the HTM model can consistently be optimized for financial markets using the GA algorithm*".

$$\exists m \forall v, i \left(P_{2b}(m, v, i) \rightarrow C_{2b}(m)\right) \quad m \in M,\ v \in V,\ i \in I \qquad \text{(H.2b)}$$

where,
P_{2b} = *"GA-optimized HTM model m yields a positive average return on validation dataset v in iteration i*".
C_{2b} = *"HTM model m can consistently be optimized for financial markets using the GA algorithm*".

These hypotheses are directly associated with research objective (RO.2) in chapter 1.4 and research question (RQ.2) in chapter 1.1.

11.3 Hypothesis 3

According to the theory in chapter 6, a model that performs well, not only on a validation dataset, but also on a test dataset, consisting of previously unseen data, shows good generalization abilities. Furthermore, as described by the theory, cross-validation is generally accepted as a good method for assessing a model's generalization ability over multiple datasets. It is therefore hypothesized, that HTM models trained, validated and tested using cross-validation, and which also yield positive returns in both the validation- and test datasets, implies that they generalize well to previously unseen data (H.3a).

"*For some HTM model and all validation- and test datasets, if the HTM model yields a positive average return on the datasets, then the HTM model generalizes well to novel data*".

$$\exists m \forall v, t \left(P_{3a}(m, v, t) \rightarrow C_{3a}(m) \right) \qquad m \in M,\ v \in V,\ t \in T \tag{H.3a}$$

where,
P_{3a} = *"HTM model m yields a positive average return on validation dataset v and test dataset t*".
C_{3a} = *"HTM model m generalizes well to novel data*".

As with hypothesis two, it must be asserted that the GA-optimized HTM models generalize well every time they are exposed to the datasets. Therefore, the following hypothesis is conjectured (H.3b):

"*For some HTM model and all validation- and test datasets in all iterations, if the HTM model yields a positive average return on the datasets, then the HTM model generalizes well to novel data*".

$$\exists m \forall v, t, i \left(P_{3b}(m, v, t, i) \rightarrow C_{3b}(m) \right) \qquad m \in M,\ v \in V,\ t \in T,\ i \in I \tag{H.3b}$$

where,
P_{3b} = *"HTM model m yields a positive average return on validation dataset v and test dataset t in iteration i*".
C_{3b} = *"HTM model m consistently generalizes well to novel data*".

These hypothesis corresponds to research objective (RO.3) in chapter 1.4 and research question (RQ.3) in chapter 1.1.

11.4 Hypothesis 4

Artificial neural networks (ANNs) are by far the most prevalent technique used to create predictive models, according to the related work presented in chapter 10. The related work also shows multiple cases where neural networks have been optimized using the genetic algorithm. Chapter 7 details the inner workings of neural networks, where they are described as universal function approximators. As with the HTM models, artificial neural networks have a number of parameters that can be optimized using the genetic algorithm. When it comes to time series, two specific types of neural networks are commonly used to capture the time-dependency in the dataset; tapped-delay neural networks and recurrent neural networks. Since neural networks are considered to be the benchmark technology for predictive models, they have been used as such in this theses. In order to compare the performance of the HTM technology against the neural network technology, the HTM models are compared to recurrent neural networks. The following null hypothesis states that HTM models consistently outperform artificial neural networks (H.4).

"*For some GA-optimized HTM model and some GA-optimized ANN model, on all test datasets in all iterations, if the HTM model yields a higher positive average return on the datasets than the ANN model, then the HTM model consistently outperforms the ANN model*".

$$\exists m, n \forall t, i \left(P_4(m,n,t,i) \rightarrow C_4(m,n)\right) \quad m \in M,\ n \in N,\ t \in T,\ i \in I \qquad \text{(H.4)}$$

where,
P_4 = *"GA-optimized HTM model m yields a higher positive average return on test dataset t in iteration i than GA-optimized ANN model n*".
C_4 = *"HTM model m consistently outperforms ANN model n*".

These hypotheses are directly associated with research objective (RO.4) in chapter 1.4 and research question (RQ.4) in chapter 1.1.

This concludes the definition of the hypotheses used to assess the success of the study conducted in this theses. The next chapter details the actual method used to implement predictive models for financial time series.

12 Method

This chapter brings together the theory described in previous chapters to develop a method for evolving HTM-based and ANN-based trading models for the E-mini S&P 500 futures market. The initial subchapter describes the source used in the data acquisition process, followed by a detailed presentation of the preprocessing and data transformation approach utilized in order to obtain feature vectors for training and validating the trading models. The genetic representation of the HTM-based and ANN-based problems are presented in separate subchapters. The experimental design is introduced in the next chapter (chapter 13).

12.1 Data Acquisition

Multiple sources were investigated for obtaining historical data for the E-mini S&P 500 futures market. The Chicago Mercantile Exchange (CME) naturally offers historical data for all derivatives traded on their Globex electronic platform, but requires a subscription fee. Other leading, commercial suppliers of financial market data are the major news agencies, such as Reuters and Bloomberg, CQG, EODdata and Morningstar Commodity Data (previously Logical Information Machines) which mostly provides commodity data.

The two most prevalent sources of free financial data are Google Finance and Yahoo Finance. In general, the free provides provide end of day data for each trading day dating back a year or two. TradingBlox provides free end of day data for futures and Forex contracts going back ten years. They roll the front month contract forward to the next contact period seven days prior to expiry. Due to the large amount of available end of day data, an initial experimentation was conducted using TradingBlox's data. Although, this approach was later abandoned for the benefit of obtaining a larger amount of data points using intra-day data with a minute resolution.

When it comes to intra-day data, finding free providers is hard, but SlickCharts does provide such a service for four E-mini markets, including the E-mini S&P 500 futures market, with a backlog of the previous five trading days. Since an intra-day resolution was desired for this research work, the SlickCharts website was scraped during the period of one year. Although, the data quality obtained from free providers is far inferior to the commercial providers. The SlickCharts data is no exception to the rule and, hence, a substantial part of the data had to be discarded due to discontinuity in the data discovered during the preprocessing stage. Nevertheless, two months worth of continuous data was salvaged for the September 2011 contract (5th July - 2nd September 2011). Using an intra-day resolution, this is still a substantial amount of data points. The decision was made to use the September contract data for training, validation and testing purposes using a modified cross-validation approach.

SlickCharts creates one comma-separated file for each contract per trading day containing one row for each transaction. Each row is structured according to Table 12.1.

Table 12.1 SlickCharts File Format

Field	Description	Example
1	E-mini Futures Code	'ES' for the E-mini S&P 500 Futures market
2	Expiration Year-Month	'2011-09' for the September 2011 contract
3	Trade Date	'7/5/2011' for a transaction made on the 5th of July 2011
4	Trade Time	'03:00:00 PM' for a transaction made at exactly 15:30:00
5	Price	'133600' for a price of 1336.00
6	Volume	'354' for 354 traded contracts

A sample row, using the information as described in the table above, would be:

```
ES,2011-09,7/5/2011,03:30:00 PM,133600,354
```

12.2 Aggregation

MATLAB™ (R2011a) was used for preprocessing the data, including aggregation, data cleaning and feature extraction. The first preprocessing task was to load the comma-separated files, one per trading day, into MATLAB™ and combine them into a continuous time series. As mentioned above, only one contract period was considered due to discontinuities in the downloaded data sets (i.e. one time series for the September 2011 quarterly contact). The time series was then aggregated, or resampled, into one-minute intervals, referred to as *bars* in the following text.

For each one-minute bar, the price level at the beginning and end of the one-minute period was determined. Furthermore, the highest and lowest price level during the one-minute period was calculated, together with the total volume of contracts traded during the one-minute period. Each bar, therefore, contained the open-, high-, low- and closing price for each respective one-minute period together with the trading volume for that period. These five scalars are commonly referred to using the abbreviation *OHLCV* (Open High Low Close Volume) as described in chapter 4.

Figure 12.1 depicts a *barplot*, which shows the price bars plotted along a time axis. The topmost point in a bar (vertical line) is the highest price for the period and the lowest point, the lowest price. A small horizontal line protruding to the left of the bar depicts the opening price for the period and a line protruding to the right, the closing price.

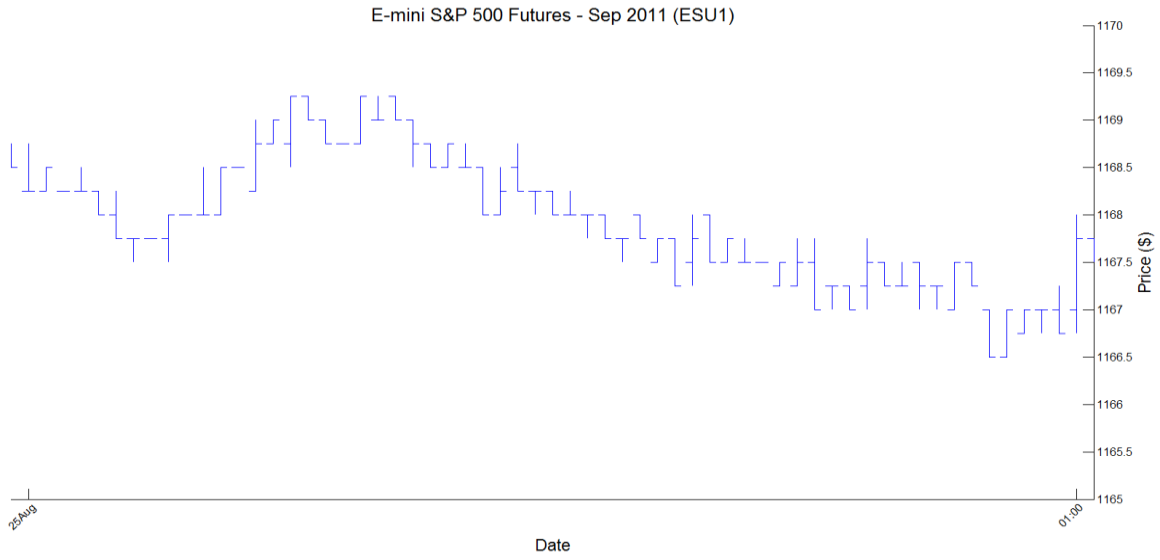

Figure 12.1 Barplot of the E-mini S&P 500 futures index market September 2011 contract (first hour of 25th of August)

A plot of *Japanese Candlesticks*, a more popular variation of bars, is shown in Figure 12.2. Similar to bars, the highest and lowest points of each vertical line constitutes the highest and lowest prices for the period. The topmost and bottommost boundaries of the box determine the opening and closing prices. If the closing price is higher than the opening price, the box is empty (white), whereas if the closing price is lower than the opening price, the box is opaque (blue).

Figure 12.2 Candle stick plot of the E-mini S&P 500 futures index market September 2011 contract (first hour of 25th of August)

Both types of plots are usually combined with a bar graph, in which the height of each bar relates to the traded volume for the period. Figure 12.3 depicts the bar graph associated with the bar plot and the candle stick plot above.

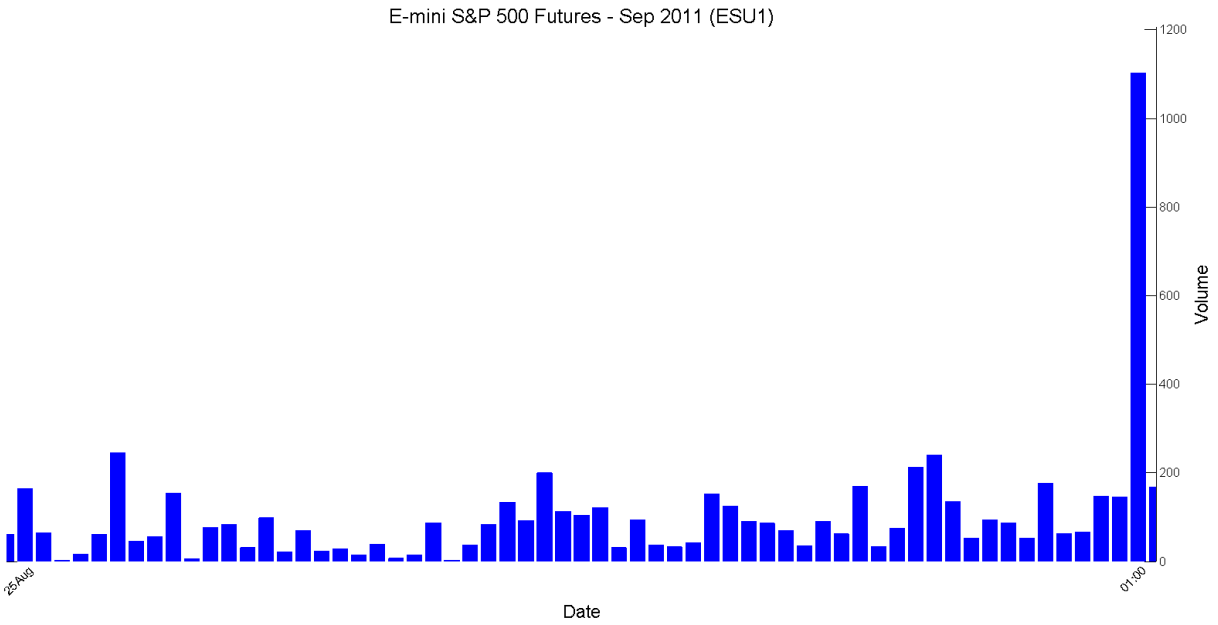

Figure 12.3 Volume plot of the E-mini S&P 500 futures index market September 2011 contract (first hour of 25th of August)

In order to maintain a continuous time series, any one-minute periods for which there was no trading activity, the closing price of the previously existing bar was used to create the missing bar, i.e. the previous closing price was used as the opening-, highest-, lowest- and closing price for the period, and the trading volume was set to zero. This approach of handling missing data points can affect the results if the time series contains a lot of missing data points, although this was not the case with the selected time series since trading activity was intense during the chosen trading period.

12.3 Feature Extraction

A set of 12 technical indicators were extracted from the aggregated data in order to populate the feature vectors for the classification models. The indicators, together with their descriptions, plots and mathematical formulas can be found in Appendix A. Table 12.2 summarizes the technical indicators together with brief descriptions of them.

Table 12.2 Technical Indicators

Indicator	Description
PPO(12,26)	*Percentage Price Oscillator*, calculated by subtracting the 26-minute exponential moving average (EMA) from the 12-minute EMA, then dividing the difference with the 26-minute EMA and multiplying the result by 100.
EMA(9)PPO	*PPO Signal Line*, calculated as the 9-minute EMA of PPO.
PPOHIST(12,26,9)	*PPO Histogram*, calculated by subtracting the PPO signal line from the PPO indicator.
RSI(14)	*Relative Strength Index*, calculated using a 14-minute period.
%R(14)	*William's %R*, calculated using a 14-minute period.
NVI(10)	*Normalized Volatility Indicator*, calculated using a 10-minute period.
NVI(20)	*Normalized Volatility Indicator*, calculated using a 20-minute period.
CMF(20)	*Chaikin Money Flow*, calculated using a 20-minute period.
%B(20,2)	*Bollinger Band %B*, calculated using a 20-minute period for the Bollinger Bands with a standard deviation of 2.
ROC(12)	*Rate Of Change*, calculated using a 12-minute period.
%K(14)	*Fast Stochastic Oscillator*, calculated using a 14-minute period.
%D(3)	*Fast Stochastic Oscillator Signal Line*, calculated using the 3-minute simple moving average (SMA) of the %K indicator.

Before the indicators were presented to the models, they were normalized (or more correctly, standardized) by subtracting the mean from each indicator and dividing by their corresponding standard deviation, hence yielding a distribution with zero mean and unit deviation. This batch processing procedure of the training data does not seriously affect an online, real-time system as long as the same mean and standard deviation values are used during the real-time calculations.

12.4 Dataset Partitioning

One major difference between non-sequential and sequential (ordered) datasets is that the order in which each data point is presented to a model needs to be preserved for sequential data (such as financial time series). This limits the use of common techniques such as ordinary cross validation. One viable solution is to partition the data into three continuous but disjoint sets; a training-, validation- and test set. The models are then trained on the training set, using the validation set to guide the search for an optimal

solution. Once a candidate model has been found, its generalization behavior can be approximated by testing its performance on the test set. Although, the downside of this approach is that fewer data points are available for training the models. Cross validation, on the other hand, permits the training and validation sets to be combined by partitioning the combined dataset into *k* folds, where *k-1* folds are used to train the model and the remaining fold is used to validate the model. This process is then repeated *k* times so that each fold is used as the validation set once, and only once. An additional benefit of using *k*-fold cross validation is that the variance in the dataset is reduced. All these concepts were described in chapter 6.

In order to reap the benefits of using a cross validation approach, a modified *k*-fold cross validation technique was employed. This was accomplished by creating a large enough initial window of size *N* to be used as the initial training set, followed by partitioning the remaining dataset into *k* folds of size *M*. The cross validation procedure then commenced by training the model on the data points contained within the training window and validating the model on the closest fold *ahead of the training window*. This was done for all individuals through a number of generations using the genetic algorithm. The fittest individual from the final generation was then tested on the closest fold *ahead of the validation fold*. Next, the window of size *N* was rolled forward *M* data points to include the first fold in the training set and, hence, omitting the oldest *M* data points of the initial window. The model was then trained on the second window of size *N* and validated on the next fold ahead of the training window. Once again, this was done for all individuals through a number of generations using the genetic algorithm and the fittest individual from the final generation was tested on the closest fold *ahead of the validation fold*. This procedure was repeated until *k-1* folds had been used as the validation set once and *k-1* folds had been used as the test set once. The performance measure obtained from all *k-1* validations was then averaged to yield an overall performance measure for the model, similar to normal *k*-fold cross validation. Similarly, the performance measure obtained from all *k-1* test folds was averaged to yield an overall performance measure for the fittest models on the test set. Finally, since the genetic algorithm is non-deterministic, the whole procedure was repeated three times and a final averaged performance measure was obtained by averaging the averaged performance measures from all three individual iterations.

The training, validation and test datasets are shown in Figure 12.4 (top). The figure also illustrates the modified *k*-fold procedure, where the initial training window (training window 01) is show to the far left in the figure, followed by validation fold 01 and test fold 01. Validation and test fold 02-04 are also depicted in the figure and the range of training windows 02-04 are shown below the top figure. The initial training window is also marked as 'Fold00' and the actual folds used for validation are marked as 'Fold01' to 'Fold04' whereas the folds used for testing are marked as 'Fold02' to 'Fold05'. In other words, *k* was set to 5. The initial training window contains 16 trading day's worth of data points and the validation folds contain over a week's worth of data points with a minute resolution. The red patch, one day's worth of data points, to the left in the figure shows the buffer used to get the moving average calculations going for the technical indicators. The top plot shows the price action and the bottom plot the corresponding volume.

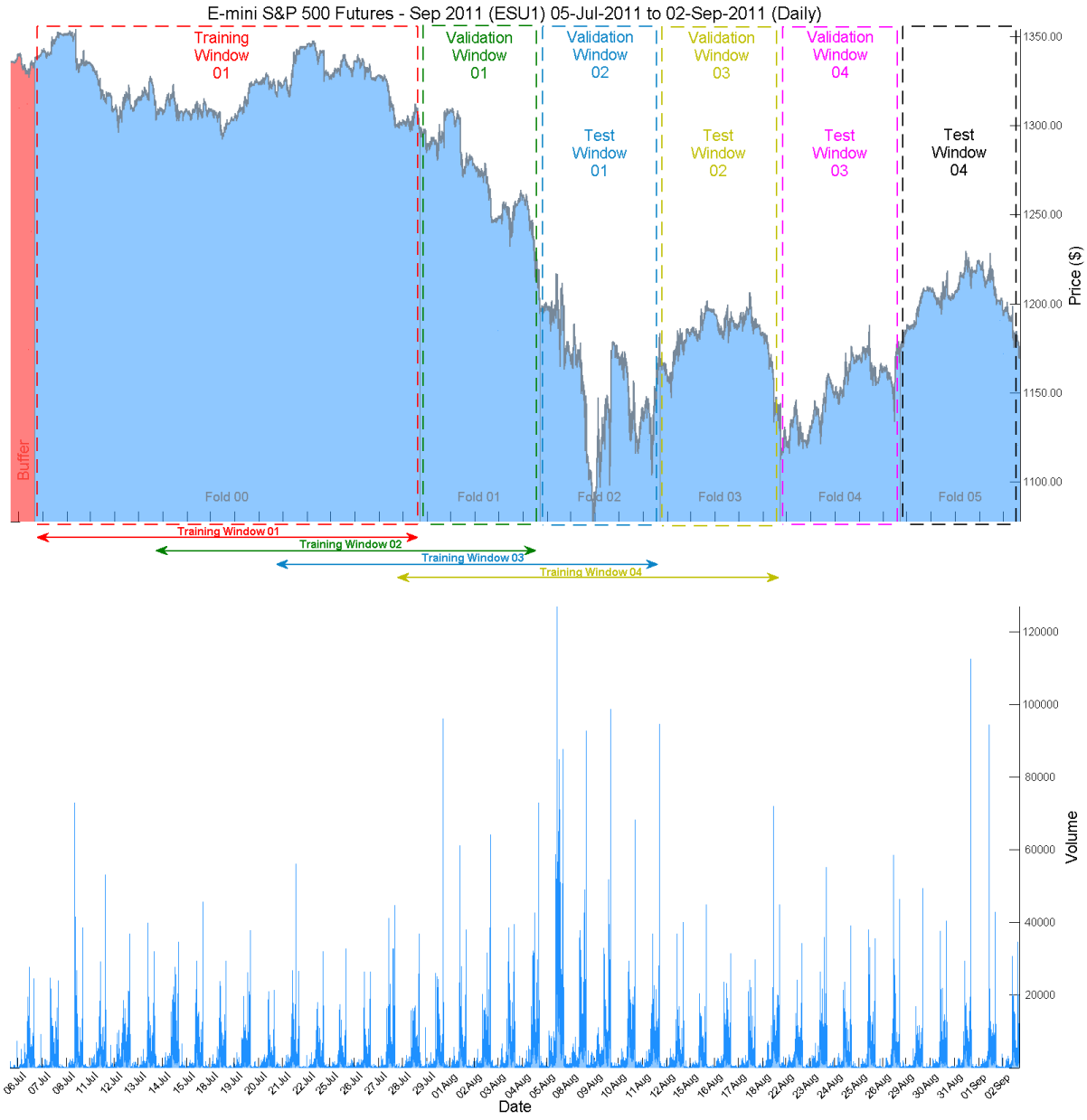

Figure 12.4 The dataset for the E-mini S&P 500 (September 2011 contract)

The figure also reveals some information about the data. All folds contain considerable volatility in the price action; Fold 00 includes short-term bearish, bullish and horizontal trends, Fold 01 contains a deep downtrend, Fold 02 is highly volatile and contains major up and down trends, Fold 03 is mostly horizontal with a final downtrend, Fold 04 has an uptrend, and Fold 05 has an initial uptrend followed by a final downtrend.

12.5 Class Definition and Trading Strategy

Considering the time periods used for the technical indicators, a time horizon of ten minutes was chosen for the predictive task. In other words, based on the current indicator readings, a model should predict the direction of the market as accurately as possible at a point in time ten minutes into the future.

A binary classification task was defined in order to supervise the learning of the models. One of the two target classes was defined as *"the market will rise by at least two ticks at the end of the next ten minute period"* and the other class as the opposite outcome *"the market will not rise by at least two ticks at the end of the next ten minute period"*. The class definitions are reiterated in Table 12.3. One tick is a 0.25 move in the E-mini S&P 500 futures index, and is worth $12.50 per contract.

Table 12.3 Class Definitions

Class	Description
0	The market will rise by at least 2 ticks at the end of the next ten minute period
1	The market will *not* rise by at least 2 ticks at the end of the next ten minute period

A *buy-and-hold* trading strategy was then derived from the binary classification. In other words, if a model predicts that the market will rise by at least two ticks at the end of the next ten minute period, a trading decision was made to buy 1 contract, hold on to it for ten minutes and sell it at the current price level at the end of the next ten minute period. If the market was predicted *not* to rise by at least two ticks at the end of the next ten minute period, a trading decision was made to remain idle. The trading strategy is reiterated in Table 12.4.

Table 12.4 Trading Strategy

Class	Description
0	Buy 1 contract, hold on to it for ten minutes, then sell it at the current price level at the end of the next ten minute period.
1	Remain idle.

12.6 Performance Measure

Multiple choices were considered for a performance measure for the models. Initially the accuracy or the precision of a model was investigated. The accuracy was discarded since it proved to be unsuitable for this application. Only true positives and false positives have a direct impact on financial gain or loss, since both result in contracts being bought. Since the accuracy for a predictive classification model is computed by dividing the sum of the true positives and true negatives with the total sum of all classification attempts, it fails to relate true positives to false positives and, hence, is a poor measure within the given context. Next, the precision was investigated. It does indeed relate true positives to false positives by dividing the true positives with the total sum of all positives. At first this seemed to be the obvious choice, but after further consideration it failed to incorporate the actual profit made by a model. Therefore, one possible solution was to combine the precision with the profit-or-loss (PNL) of a model. This also proved to be a futile attempt.

A final decision was made to just use the PNL as the performance measure for a model. This makes sense, since when push comes to shove, it all boils down to finding models that make as large a profit as possible. In order to penalize too aggressive models, the total trading costs were deducted to yield the final PNL for a model. The costs incurred by a model were defined as $3 per contract and round trip, i.e. $3 for every true positive or false positive. This performance measure was also used as the fitness function while optimizing the models with the genetic algorithm.

12.7 Model Optimization

Both the HTM and the ANN models were optimized using the genetic algorithm by mapping model parameters into chromosomes. As described in the previous chapter, the PNL was used as the fitness function for validating each individual. Although, due to the possibility of negative PNLs and since a relative fitness value of an individual is normally computed as the fitness of the individual divided by the overall fitness of the population, the fitness score of an individual was normalized between 0 and 1 by subtracting each individual's PNL by the smallest PNL in the current generation and dividing the difference by the range of the PNL in the current generation, i.e. by using Equation 6.2 in Chapter 6).

Since the HTM framework, specifically the freely available research version (NuPIC™ version 1.7.1), uses Python 2.5 as its scripting language, the choice was made to implement the genetic algorithm logic and the neural networks in Python 2.5. To aid in the process, the following additional Python packages were referenced; Numpy 1.6.1 (MATLAB™ and matrix algebra support), Matplotlib 1.1.0 (MATLAB™ plotting functionality), PIL 1.1.7 (additional plotting functionality), Scipy 0.10.1 (scientific calculation support), Pybrain 0.3.1 (machine learning support, including neural networks) and Pyevolve 0.5 (genetic algorithm support).

13 Experiments

The experiments include the mapping of model parameters into chromosomes, followed by evolving HTM-based and ANN-based models using the genetic algorithm. The best resulting individuals were then used to test the models' performance on the test dataset.

13.1 Evolving the HTM Models

An HTM network has a number of different parameters than can be optimized. The most important parameters, tabulated in Table 13.1, were chosen for the optimization task.

The parameters for network *Topology*, *Coincidences*, *Max Distance*, *Sigma* and *Groups* were detailed in Chapter 8.

Table 13.1 HTM Parameters

Parameter	Description
General Parameters	
Topology	Network topology, i.e. number of layers, including number of nodes per layer
Spatial Pooler	
Coincidences	The maximum number of spatial patterns that can be learned
Max Distance	Maximum Euclidean distance for clustering spatial patterns to centroids
Sigma	Sigma to be used in the radial-basis function in Gaussian inference mode
SP Algorithm	Spatial Pooler's inference algorithm {Gaussian, K^{th} Root Product}
Temporal Pooler	
Groups	The maximum number of temporal patterns that can be learned
Complexity Model	Complexity of the sequence model built by the sequencer [0 - 1.0]
Window Count	Number of windows over which the sequencer will build a sequence model
Window Length	Number of samples in each window where the sequencer will build a model
TP Algorithm	Temporal Pooler's inference algorithm {Max Prop, Sum Prop, TBI}
Transition Memory	How far back in time to look for temporal transitions (to smooth temporal jitter)

The *SP Algorithm* parameter determines the Spatial Pooler's inference algorithm; *Gaussian* or *K^{th} Root Product*. Gaussian inference, which was also described in Chapter 8, uses the *Sigma* parameter to determine the standard deviation for the Gaussian curve used in the Radial Basis Function. The *K^{th} Root Product* setting forms the geometric mean of the probabilities associated with the temporal groups of child nodes to infer which coincidence they belong to in the parent node. The two parameters *Sigma* and *SP Algorithm* only affect the behavior of the Spatial Pooler in inference mode.

In Chapter 8, the forming of temporal groups in the Temporal Pooler was described as creating a normalized transition matrix of coincidences and visualized as a directed graph, which was partitioned into maximally coherent temporal groups using a greedy algorithm. During inference, the group containing the currently sensed coincidence was then chosen as the most likely group. This type of inference is called *Flash Inference*, since the inferred temporal group is based solely on the currently sensed input (coincidence). Another type of inference, called *Time Based Inference (TBI)*, bases its decision on the current input (coincidence) and a number of previously sensed inputs (coincidences). The *TP Algorithm* parameter determines the Temporal Pooler's inference algorithm; *Max Prop*, *Sum Prop* or *TBI*. The two settings *Max Prop* and *Sum Prop* are both *Flash Inference* algorithms whereas the *TBI* setting uses *Time Based Inference*.

The *TP Algorithm* parameter only affects the behavior of the Temporal Pooler in inference mode, but relies on four parameters that determine the type of information that is stored in the Temporal Pooler during learning. The *Transition Memory* parameter is used for *Flash Inference* and determines the history length of coincidence transitions that are stored in the Temporal Pooler. This information is only used to smooth temporal jitter (to reduce temporal noise).

The three parameters; *Complexity Model*, *Window Count* and *Window Length* have to do with *Time Based Inference (TBI)* and work together to create a higher order *Markov Chain* during learning. The normalized transition matrix and directed graph, described in Chapter 8, from which temporal groups were created are in essence a Markov Chain. A Markov Chain has a number of finite states (the vertices in the directed graph) and a number of transition probabilities (the edges in the directed graph). In a traditional Markov Chain, the probability of transitioning from one state (vertex) to another only depends on the current state (vertex) and not on the history of prior states leading up to the current state. Furthermore, the probability distribution for transitioning from one state to another is the same for each state. The mathematical term for this is *Independent and Identically Distributed (IID)* random variables. In other words, the vertices in the directed graph can be seen as a random variables with identically distributed transition probabilities and are independent of the outcome of other random variables (vertices). This property is called the *Markov Property*.

Relating the theory of Markov Chains to the Temporal Pooler, the normalized transition matrix (and normalized directed graph), that describes the probability transitions from one coincidence to the next is a first order Markov Chain (traditional Markov Chain). It is a first order Markov Chain since it describes the transition probabilities from one state to the immediate next state. A higher order Markov Chain is an extension of the traditional Markov Chain. It remembers more than one transition from each state. Using a second order Markov Chain, the Temporal Pooler stores two transitions from each coincidence, for example C(t-2) to C(t-1) to C(t) where *t* refers to the current time step, *t-1* refers to the previous time step and *t-2* refers to two time steps ago relative to the current time step. For higher order Markov Chains even more time steps are stored. For Flash Inference, as described in Chapter 8, the first order Markov Chain (transition matrix and directed graph) is partitioned into maximally coherent temporal groups, from which one group is inferred based solely on the currently sensed coincidence. Since no sequence information is preserved amongst the coincidences in each group, the temporal groups actually constitute zero order Markov Chains. This is perfectly fine for Flash Inference. For *Time Based Inference (TBI)*, sequential information needs to be preserved and therefore the *Time Based Inference (TBI)* algorithm relies on sequential information being stored in each group. This learning behavior is managed by a *Sequencer* in the Temporal Pooler and is configured with the three parameters *Complexity Model*, *Window Count* and *Window Length*. The *Complexity Model* parameter ranges from 0.0 to 1.0 and determines the temporal order complexity of the learned temporal groups. A higher setting for this parameter results in more sequential information being stored in the transition matrix. The *Window Count* parameter determines the number of stages the Temporal Pooler uses to look for sequential information. A higher setting results in more sequential information being extracted from the data. The *Window Length* parameter specifies the number of samples (coincidences) the Temporal Pooler uses in each window to discover sequential information. In other words, the temporal statistics are accumulated for these many coincidences for each window.

Besides the configurable HTM parameters described above, the HTM networks were created using a *Fan In* link type for all nodes (equivalent to a fully connect feed-forward neural network). Furthermore, three different types of topmost nodes are available in NuPIC version 1.7.1; *SVMClassifierNode*, *KNNClassifierNode* and *Zeta1TopNode*. The SVMClassifierNode implements the *Support Vector Machine* algorithm, the KNNClassifierNode uses the *k Nearest Neighbors* algorithm and the Zeta1TopNode implements the *Naïve Bayes* algorithm. The choice was made to use the Zeta1TopNode as the topmost node. All other nodes implement Numenta's proprietary *SpatialPooler* and *TemporalPooler* algorithms.

When it comes to the network topology, an HTM requires the node count in lower layers to be evenly divisible by the node count in higher layers. This places a constraint on the possible combinations of network topologies. Since there is one node per input feature in the bottommost layer, the bottom layer always contains 12 nodes in this study. With this in mind, and limiting the highest topology order to 4 due to memory and time constraints, 64 possible network topologies where identified as listed in Table 13.2.

Table 13.2 HTM Network Topologies (topmost classifier node not included)

Topology	Number of Nodes (Layer)				Topology	Number of Nodes (Layer)			
[12,1]	12 (I),	1 (II)			[12,4,2,1]	12 (I),	4 (II),	2 (III),	1 (IV)
[12,2]	12 (I),	2 (II)			[12,4,2,2]	12 (I),	4 (II),	2 (III),	2 (IV)
[12,3]	12 (I),	3 (II)			[12,4,4,1]	12 (I),	4 (II),	4 (III),	1 (IV)
[12,4]	12 (I),	4 (II)			[12,4,4,2]	12 (I),	4 (II),	4 (III),	2 (IV)
[12,6]	12 (I),	6 (II)			[12,4,4,4]	12 (I),	4 (II),	4 (III),	4 (IV)
[12,12]	12 (I),	12 (II)			[12,6,1,1]	12 (I),	6 (II),	1 (III),	1 (IV)
[12,1,1]	12 (I),	1 (II),	1 (III)		[12,6,2,1]	12 (I),	6 (II),	2 (III),	1 (IV)
[12,2,1]	12 (I),	2 (II),	1 (III)		[12,6,2,2]	12 (I),	6 (II),	2 (III),	2 (IV)
[12,2,2]	12 (I),	2 (II),	2 (III)		[12,6,3,1]	12 (I),	6 (II),	3 (III),	1 (IV)
[12,3,1]	12 (I),	3 (II),	1 (III)		[12,6,3,3]	12 (I),	6 (II),	3 (III),	3 (IV)
[12,3,3]	12 (I),	3 (II),	3 (III)		[12,6,6,1]	12 (I),	6 (II),	1 (III),	1 (IV)
[12,4,1]	12 (I),	4 (II),	1 (III)		[12,6,6,2]	12 (I),	6 (II),	6 (III),	2 (IV)
[12,4,2]	12 (I),	4 (II),	2 (III)		[12,6,6,3]	12 (I),	6 (II),	6 (III),	3 (IV)
[12,4,4]	12 (I),	4 (II),	4 (III)		[12,6,6,6]	12 (I),	6 (II),	6 (III),	6 (IV)
[12,6,1]	12 (I),	6 (II),	1 (III)		[12,12,1,1]	12 (I),	12 (II),	1 (III),	1 (IV)
[12,6,2]	12 (I),	6 (II),	2 (III)		[12,12,2,1]	12 (I),	12 (II),	2 (III),	1 (IV)
[12,6,3]	12 (I),	6 (II),	3 (III)		[12,12,2,2]	12 (I),	12 (II),	2 (III),	2 (IV)
[12,6,6]	12 (I),	6 (II),	6 (III)		[12,12,3,1]	12 (I),	12 (II),	3 (III),	1 (IV)
[12,12,1]	12 (I),	12 (II),	1 (III)		[12,12,3,3]	12 (I),	12 (II),	3 (III),	3 (IV)
[12,12,2]	12 (I),	12 (II),	2 (III)		[12,12,4,1]	12 (I),	12 (II),	4 (III),	1 (IV)
[12,12,3]	12 (I),	12 (II),	3 (III)		[12,12,4,2]	12 (I),	12 (II),	4 (III),	2 (IV)
[12,12,4]	12 (I),	12 (II),	4 (III)		[12,12,4,4]	12 (I),	12 (II),	4 (III),	4 (IV)
[12,12,6]	12 (I),	12 (II),	6 (III)		[12,12,6,1]	12 (I),	12 (II),	6 (III),	1 (IV)
[12,12,12]	12 (I),	12 (II),	12 (III)		[12,12,6,2]	12 (I),	12 (II),	6 (III),	2 (IV)
[12,1,1,1]	12 (I),	1 (II),	1 (III),	1 (IV)	[12,12,6,3]	12 (I),	12 (II),	6 (III),	3 (IV)
[12,2,1,1]	12 (I),	2 (II),	1 (III),	1 (IV)	[12,12,6,6]	12 (I),	12 (II),	6 (III),	6 (IV)
[12,2,2,1]	12 (I),	2 (II),	2 (III),	1 (IV)	[12,12,12,1]	12 (I),	12 (II),	12 (III),	1 (IV)
[12,2,2,2]	12 (I),	2 (II),	2 (III),	2 (IV)	[12,12,12,2]	12 (I),	12 (II),	12 (III),	2 (IV)
[12,3,1,1]	12 (I),	3 (II),	1 (III),	1 (IV)	[12,12,12,3]	12 (I),	12 (II),	12 (III),	3 (IV)
[12,3,3,1]	12 (I),	3 (II),	3 (III),	1 (IV)	[12,12,12,4]	12 (I),	12 (II),	12 (III),	4 (IV)
[12,3,3,3]	12 (I),	3 (II),	3 (III),	3 (IV)	[12,12,12,6]	12 (I),	12 (II),	12 (III),	6 (IV)
[12,4,1,1]	12 (I),	4 (II),	1 (III),	1 (IV)	[12,12,12,12]	12 (I),	12 (II),	12 (III),	12 (IV)

The network topology was therefore encoded into the first six bits of a bit string chromosome since $\log_2(64) = 6$.

For the remaining parameters, a different setting per layer was desirable. Therefore, and since the maximum number of layers was set to four, the settings for each parameter were replicated four times. This solution implies that individuals with topologies consisting of fewer layers than four will contain some "don't care" bits for unused, superfluous parameters.

For the maximum number of coincidences per node, the upper limit was arbitrarily set to 4096. The rationale behind this decision was based on limited random access memory and processing power in the computer used to evolve the HTMs. Using more memory caused the python process to crash sporadically and also resulted in unfeasible run times. Since $\log_2(4096) = 12$, the next 12 bits were used to encode the coincidences setting for the first layer's nodes. An additional 12 * 3 = 36 bits were used to encode the coincidences setting for the second, third and fourth layer's nodes respectively.

By observing the range of the normalized features, it made sense to use a max distance setting within the range [0.000 and 1.000]. In order to avoid using real numbers in the chromosome, the max distance setting for the first layer nodes was encoded using the next 8 bits in the chromosome bit string. Since $\log_2(256) = 8$, using the reciprocal value of the eight bits yields the desired range [0/256, 255/256] with a 1/256 (0.0039) resolution. A max distance setting of zero means that the spatial pooler in each HTM node will create one coincidence (quantization center) for each different spatial pattern it observes, which will be discarded by the genetic algorithm if undesirable performance results are obtained for such an individual. An additional 8 * 3 = 24 bits were used to encode the max distance setting for the second, third and fourth layer's nodes respectively.

The *Sigma* parameter determines the width, in standard deviations, of the Gaussian curve in Gaussian inference mode. Since a deviation of two standard deviations (in any direction from the center of the Gaussian hyper-surface) is quite large, this was deemed as a good choice for an upper limit. Similar to the max distance parameter, the Sigma parameter was encoded with $\log_2(256) = 8$ bits to procedure the range [1/256, 256/256]. This value was then multiplied by 2 resulting in the effective range [1/128, 256/128] with a 1/128 (0.0078) resolution, i.e. a Sigma ranging from 0.0078 up to 2.0000. An additional 8 * 3 = 24 bits were used to encode the Sigma setting for the final three layer's nodes.

The Spatial Pooler Algorithm parameter accepts one of two values from the set {*Gaussian*, K^{th} *Root Product*}. Therefore, $\log_2(2) = 1$ bit was used to encode the spatial pooler's inference algorithm for the first layer nodes. An additional 3 bits were used to encode the inference algorithm for the final three layers' nodes.

The same approach and rationale as with the maximum number of coincidences per node was used to set an upper limit of 4096 for the maximum number of groups per node. Therefore $\log_2(4096) * 4 = 12 * 4 = 48$ bits were used to encode the groups setting for the four layers.

The sequencer Complexity Model parameter accepts values from the range [0.000, 1.000]. Therefore the same approach as with the Max Distance setting was employed to encode the Complexity Model setting, i.e. $\log_2(256) = 8$ bits were needed per layer ($8 * 4 = 32$ in total). Using the reciprocal value of the eight bits produces the range [1/256, 256/256] with a 1/256 (0.0039) resolution.

An upper limit for the sequencer Window Count parameter needed to be chosen due to computer memory and processing power constraints. The chosen range [1, 8] needed $\log_2(8) = 3$ bits per layer (i.e. $3 * 4 = 12$ in total).

The sequencer Window Length parameter also needed an upper limit for the same reasons. The chosen range [1, 32] needed $\log_2(32) = 5$ bits per layer (i.e. $5 * 4 = 20$ in total).

The Temporal Pooler Algorithm parameter accepts one of three values from the set {*Max Prop*, *Sum Prop*, *TBI*}. Actually, there is a fourth parameter value called *Hard Coded*, but it is a special-purpose setting only used for computer vision applications and is therefore not considered in this paper. Furthermore, in the current NuPIC version (1.7.1), *Time Based Inference (TBI)* only works satisfactorily with the first layer's nodes. Therefore, one bit was used to encode either *Max Prop* or *Sum Prop* and another with was used to encode *TBI* "on" or "off". If *TBI* was "on", the temporal pooler used *Time Based Inference*. If *TBI* was "off", the temporal pooler used *Flash Inference* and used the bit setting for either *Max Prop* or *Sum Prop*. Therefore, $\log_2(4) = 2$ bits was used to encode the temporal pooler's inference algorithm per layer ($2 * 4 = 8$ in total). The value of the TBI bit was not used for the second, third and fourth layers' nodes, i.e. the TBI bit was regarded as "don't care" for these three layers.

Finally, a high setting for the Transition Memory parameter also demanded a lot of memory and processing power, so an upper limit of 8 was chosen resulting in the range [1, 8] with $\log_2(8) = 3$ bits per layer (i.e. $3 * 4 = 12$ in total).

The resulting HTM chromosome is shown in Table 13.3.

Table 13.3 HTM Chromosome

Bits	Description
[0:5]	Network topology, ranging from 1 to 64.
[6:17]	Coincidences for layer 1, ranging from 1 to 4096.
[18:29]	Coincidences for layer 2, ranging from 1 to 4096.
[30:41]	Coincidences for layer 3, ranging from 1 to 4096.
[42:53]	Coincidences for layer 4, ranging from 1 to 4096.
[54:61]	Max Distance for layer 1, ranging from 0 to 0.9961 (0/256 to 255/256).
[62:69]	Max Distance for layer 2, ranging from 0 to 0.9961 (0/256 to 255/256).
[70:77]	Max Distance for layer 3, ranging from 0 to 0.9961 (0/256 to 255/256).
[78:85]	Max Distance for layer 4, ranging from 0 to 0.9961 (0/256 to 255/256).
[86:93]	Sigma for layer 1, ranging from 0.0078 to 2.0 (1/128 to 256/128).
[94:101]	Sigma for layer 2, ranging from 0.0078 to 2.0 (1/128 to 256/128).
[102:109]	Sigma for layer 3, ranging from 0.0078 to 2.0 (1/128 to 256/128).
[110:117]	Sigma for layer 4, ranging from 0.0078 to 2.0 (1/128 to 256/128).
[118]	Spatial Pooler Algorithm layer 1 {0: Gaussian, 1: K^{th} Root Product}.
[119]	Spatial Pooler Algorithm layer 2 {0: Gaussian, 1: K^{th} Root Product}.
[120]	Spatial Pooler Algorithm layer 3 {0: Gaussian, 1: K^{th} Root Product}.
[121]	Spatial Pooler Algorithm layer 4 {0: Gaussian, 1: K^{th} Root Product}.
[122:133]	Groups for layer 1, ranging from 1 to 4096.
[134:145]	Groups for layer 2, ranging from 1 to 4096.
[146:157]	Groups for layer 3, ranging from 1 to 4096.
[158:169]	Groups for layer 4, ranging from 1 to 4096.
[170:177]	Model Complexity for layer 1, ranging from 0.0039 to 1.0 (1/256 to 256/256).
[178:185]	Model Complexity for layer 2, ranging from 0.0039 to 1.0 (1/256 to 256/256).
[186:193]	Model Complexity for layer 3, ranging from 0.0039 to 1.0 (1/256 to 256/256).
[194:201]	Model Complexity for layer 4, ranging from 0.0039 to 1.0 (1/256 to 256/256).
[202:204]	Sequencer Window Count for layer 1, ranging from 1 to 8.
[205:207]	Sequencer Window Count for layer 2, ranging from 1 to 8.
[208:210]	Sequencer Window Count for layer 3, ranging from 1 to 8.
[211:213]	Sequencer Window Count for layer 4, ranging from 1 to 8.
[214:218]	Sequencer Window Length for layer 1, ranging from 1 to 32.
[219:223]	Sequencer Window Length for layer 2, ranging from 1 to 32.
[224:228]	Sequencer Window Length for layer 3, ranging from 1 to 32.
[229:233]	Sequencer Window Length for layer 4, ranging from 1 to 32.
[234:235]	Temporal Pooler Algorithm layer 1 {0: Max Prop, 1: Sum Prop, 2: TBI On, 3: TBI Off}
[236:237]	Temporal Pooler Algorithm layer 2 {0: Max Prop, 1: Sum Prop, 2: TBI On, 3: TBI Off}
[238:239]	Temporal Pooler Algorithm layer 3 {0: Max Prop, 1: Sum Prop, 2: TBI On, 3: TBI Off}
[240:241]	Temporal Pooler Algorithm layer 4 {0: Max Prop, 1: Sum Prop, 2: TBI On, 3: TBI Off}
[242:249]	Transition Memory for layer 1, ranging from 1 to 8.
[250:257]	Transition Memory for layer 2, ranging from 1 to 8.
[258:265]	Transition Memory for layer 3, ranging from 1 to 8.
[266:273]	Transition Memory for layer 4, ranging from 1 to 8.

With the chromosome representation and the fitness function in place, the remaining parameter settings for the genetic algorithm are tabulated in Table 13.4.

Table 13.4 HTM Genetic Algorithm Settings

Parameter	Setting
Genome	Binary string of length 274
Crossover	Single point
Mutator	Bit flip
Selection	Tournament-based (4 individuals per tournament)
Generations	20
Population Size	50
Crossover Rate	80%
Mutation Rate	4%
Elitism	True

As shown in the table, a bit string of length 274 was used with single point crossover, a bit flip mutator and a tournament-based selection method with 4 individuals in each tournament. The number of generations was set to 20 with a population size of 50. The rationale behind these two modest settings is because of the lack of resources. A HTM takes a considerable amount of time to train using the processing power of a single computer. Due to time constraints, higher settings were not possible during this study. A common setting was chosen for the crossover rate and the mutation rate. Elitism was used to preserve the individuals with best performance through each generation.

Each time the genetic algorithm requested a fitness score from an individual, the following sequence of events took place; firstly, an HTM network was created with the parameter settings derived from the individual's bit string pattern. Next, the HTM was trained on the data points contained within the training window and validated on the corresponding validation fold. Each training and validation step produced a performance measure (PNL subtracted by the total trading cost). At the end of each generation, each individuals' performance measure was normalized and used as the individual's fitness value by the genetic algorithm.

13.2 Evolving the ANN Models

The approach used with the HTMs was to optimize a handful of parameters and then give the HTM learning algorithms a chance to tweak the final performance by exposing them to a training set. An alternative would be to let the genetic algorithm optimize coincidences (quantization centers) and groups (with transition probabilities) in each node directly and discard the training set entirely. Likewise, for a neural network, the genetic algorithm could potentially carry-out all the optimization by tuning the topology, node connections, network weights, biases, activation functions and learning algorithms. The ANN could then simply be validated on the validation folds to assess its performance. In order to optimize the benchmark technology (ANNs) in a similar fashion to the HTM technology, the ANNs were not completely optimized by the genetic algorithm. Instead, a decision was made to just optimize the network topology, learning rate and momentum, and give the ANN learning algorithms a chance to tweak the network weights and biases by exposing them to a training dataset. This was considered a fair approach for comparison purposes.

Furthermore, according to [69] using the genetic algorithm to optimize all the ANN parameters has two drawbacks; firstly, all the local information from the targets about the error at each output node of the network is turned into just one number, the fitness, which is throwing away useful information, secondly, the gradient information is ignored, which is also throwing away useful information. With regards to this, it is beneficial to optimize the network topology with the genetic algorithm and let the ANN learning algorithm update the network weights. At first, a dual representation approach was considered with support of [55], but was abandoned considering the reasons described above. Although, the bit string chromosome for the ANN was adopted from [55].

Firstly, according to [69], using more than two hidden layers is commonly deemed as unnecessary. Therefore, a maximum of two hidden layers were used in this study. The two hidden layers were encoded as the initial bit in the bit string, i.e. with a value of '0' the second hidden layer is turned off, with a value of '1' it is turned on.

Next, the maximum number of nodes in each hidden layer was chosen to be just above the number of nodes in the input layer (12). Since $\log_2(16) = 4$, each layer was encoded using 4 bits each. A hidden layer consisting of zero nodes is pointless, therefore a bit pattern of '0000' was treated as 1 node and, hence, a bit pattern of '1111' was treated as 16 nodes.

In order to avoid using real numbers in the chromosome, the learning rate and momentum settings were encoded using the next 16 bits (8 each) in the chromosome bit string. Since $\log_2(256) = 8$, using the reciprocal value of the eight bits yields a range of [1/256, 256/256] with a 1/256 (0.0039) resolution. The final ANN bit string chromosome had a length of 25 and is tabulated in Table 13.5.

Table 13.5 ANN Chromosome

Bits	Description
[0]	Network topology, ranging from 0 to 1 (0: 1 hidden layer, 1: 2 hidden layers).
[1:4]	Number of nodes in the first hidden layer, ranging from 1 to 16.
[5:8]	Number of nodes in the second hidden layer, ranging from 1 to 16.
[9:16]	Learning rate, ranging from 0 to 255 (1/256 to 256/256).
[17:24]	Momentum, ranging from 0 to 255 (1/256 to 256/256).

The remaining ANN parameters were kept constant and are shown in Table 13.6.

Table 13.6 ANN Parameters

Parameter	Setting
Input Nodes	12 (one for each input feature)
Output Nodes	2 (one for each class)
Network Type	Fully connected recurrent network (feedback in hidden layers)
Activation Function (input layer)	Linear
Activation Function (hidden layers)	Sigmoid
Activation Function (output layer)	Softmax
Learning Algorithm	Backpropagation with gradient descent

As with the HTM, the ANN's performance measure was the PNL. The parameter settings for the ANN's genetic algorithm are shown in Table 13.7. Besides the length of the chromosome, they are identical to the settings for the HTM.

Table 13.7 ANN Genetic Algorithm Settings

Parameter	Setting
Genome	Binary string of length 25
Crossover	Single point
Mutator	Bit flip
Selection	Tournament-based (4 individuals per tournament)
Generations	50
Population Size	20
Crossover Rate	80%
Mutation Rate	4%
Elitism	True

Each time the genetic algorithm requested a fitness score from an individual, the same sequence of events as with the HTM took place. During the training process, the ANN's learning algorithm was given a chance to update the randomly initialized network weights and biases.

13.3 Evaluating Model Performance

The best individuals, one for each of the four validation sets, from the final generation were tested on the corresponding four test sets. The performance on the four validation sets were then averaged to produce a representative performance measure. The same averaging procedure was carried-out for the test sets. Lastly, the averaged performance measures from the validation- and test sets were, in turn, averaged over the three iterations to yield a final performance measure.

PART IV
Results and Conclusions

14 Results

Before the model optimization results and their performance results on the validation- and test sets are presented, an overview of the class distribution in the various datasets is given in Table 14.1 and visualized in Figure 14.1. As a reminder, class 0 is defined as "the market will rise by at least 2 ticks at the end of the next ten minute period" and class 1 is defined as "the market will not rise by at least 2 ticks at the end of the next ten minute period. As can be observed, there is a class imbalance between the two classes with more samples belonging to class 1 than class 0 in all datasets. In fact, the opportunities of making a profit are vastly outnumbered by the chances of incurring a loss.

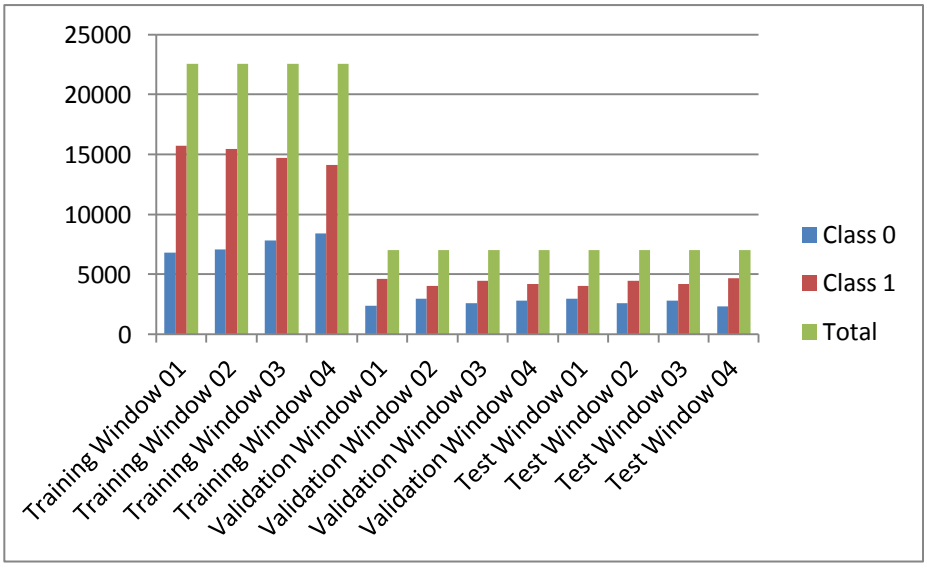

Figure 14.1 Class Distribution Per Dataset

Table 14.1 Class Distribution Per Dataset

Dataset	Class 0	Class 1	Total
Training Window 01	6801	15732	22533
Training Window 02	7091	15442	22533
Training Window 03	7838	14695	22533
Training Window 04	8403	14130	22533
Validation Window 01	2391	4631	7022
Validation Window 02	2987	4048	7035
Validation Window 03	2580	4455	7035
Validation Window 04	2819	4217	7036
Test Window 01	2987	4048	7035
Test Window 02	2580	4455	7035
Test Window 03	2819	4217	7036
Test Window 04	2334	4691	7025

14.1 HTM Optimization Results

Table 14.2 summarizes the resulting HTM parameters produced by the genetic algorithm for all datasets in all three iterations. The results show that a three-layer HTM was preferred over a two- or four-layer topology. It is also obvious that the Gaussian inference algorithm was favored over the K^{th} Root Product algorithm for the spatial pooler in every node. Furthermore, a Sigma value close to 1 standard deviation worked best together with the Gaussian inference algorithm. The temporal pooler algorithm used Time Based Inference (TBI) for half the models on average, whereas the maxProp algorithm was preferred for Flash Inference. The settings for the remaining parameters (Coincidences, MaxDistance, Groups, ModelComplexity, WindowCount, WindowLength and TransitionMemory) varied quite a lot amongst the various models. This might indicate that they are correlated with (dependent on) each other.

Table 14.2 HTM Optimization Results

Parameter	Dataset 01	Dataset 02	Dataset 03	Dataset 04
		ITERATION 1		
Topology	[12,6,3]	[12,6,6]	[12,3,3]	[12,4,2]
Coincidences	[573,511,554]	[1021,477,288]	[205,195,127]	[581,740,820]
MaxDistance	[0.004,0.009,0.011]	[0.006,0.020,0.004]	[0.067,0.005,0.011]	[0.011,0.008,0.005]
Sigma	[1.008,1.013,1.008]	[1.006,1.014,2.000]	[1.005,1.008,1.005]	[1.007,1.011,1.091]
SP Algorithm	[gauss,gauss,gauss]	[gauss,gauss,gauss]	[gauss,gauss,gauss]	[gauss,gauss,gauss]
Groups	[634,349,257]	[225,650,909]	[739,779,383]	[749,421,922]
ModelComplexity	[0.102,0.102,0.102]	[0.107,0.104,0.102]	[0.103,0.102,0.101]	[0.102,0.102,0.103]
WindowCount	[1,1,2]	[1,1,2]	[2,2,1]	[1,2,2]
WindowLength	[4,1,1]	[7,3,2]	[8,2,4]	[2,2,6]
TP Algorithm	[tbi,sumP,sumP]	[maxP,maxP,sumP]	[maxP,maxP,maxP]	[maxP,maxP,maxP]
TransitionMemory	[3,3,3]	[2,2,1]	[3,3,2]	[4,1,1]
		ITERATION 2		
Topology	[12,6,2]	[12,6,6]	[12,4,2]	[12,4,2]
Coincidences	[141,886,706]	[311,161,243]	[926,579,549]	[773,740,804]
MaxDistance	[0.004,0.006,0.004]	[0.007,0.005,0.017]	[0.005,0.006,0.029]	[0.011,0.008,0.005]
Sigma	[1.007,1.022,1.005]	[1.004,1.006,1.005]	[1.037,1.009,1.022]	[1.008,1.011,1.091]
SP Algorithm	[gauss,gauss,gauss]	[gauss,gauss,gauss]	[gauss,gauss,gauss]	[gauss,gauss,gauss]
Groups	[217,558,665]	[864,96,594]	[722,360,855]	[1005,421,922]
ModelComplexity	[0.101,0.102,0.102]	[0.104,0.104,0.101]	[0.104,0.101,0.102]	[0.102,0.102,0.103]
WindowCount	[2,2,2]	[2,2,1]	[2,1,1]	[1,2,2]
WindowLength	[2,7,8]	[8,4,4]	[7,3,1]	[2,2,6]
TP Algorithm	[tbi,maxP,sumP]	[maxP,sumP,maxP]	[tbi,sumP,sumP]	[maxP,maxP,maxP]
TransitionMemory	[2,2,1]	[4,1,1]	[2,4,4]	[4,1,3]
		ITERATION 3		
Topology	[12,6,2]	[12,4,2]	[12,4,2]	[12,6,6]
Coincidences	[964,909,118]	[212,917,334]	[768,807,845]	[325,275,127]
MaxDistance	[0.005,0.005,0.025]	[0.005,0.005,0.159]	[0.048,0.007,0.031]	[0.008,0.019,0.006]
Sigma	[1.006,1.010,1.005]	[1.007,1.015,1.125]	[1.004,1.007,1.100]	[1.008,1.025,1.027]
SP Algorithm	[gauss,gauss,gauss]	[gauss,gauss,gauss]	[gauss,gauss,gauss]	[gauss,gauss,gauss]
Groups	[482,726,844]	[937,446,478]	[872,811,723]	[742,381,93]
ModelComplexity	[0.103,0.102,0.105]	[0.101,0.106,0.102]	[0.211,0.108,0.101]	[0.101,0.102,0.137]
WindowCount	[1,2,2]	[1,1,2]	[1,1,1]	[2,2,1]
WindowLength	[3,7,8]	[2,8,8]	[4,2,6]	[4,6,5]
TP Algorithm	[sumP,maxP,sumP]	[sumP,sumP,maxP]	[tbi,sumP,sumP]	[tbi,sumP,sumP]
TransitionMemory	[3,2,3]	[1,1,2]	[3,1,1]	[2,2,2]

The performance results for the fittest HTM individuals in the final generation for all three iterations are shown in Table 14.3.

Table 14.3 HTM Performance

Dataset	TP	TN	FN	FP	Accuracy	Precision	PNL
ITERATION 1							
Training 01	471	15656	6325	76	0.716	0.861	$28309.00
Validation 01	20	4016	2083	25	0.657	0.444	$1240.00
Test 01	4	3613	2526	1	0.589	0.800	$710.00
Training 02	2669	15377	4422	60	0.801	0.978	$167750.50
Validation 02	144	3420	2386	194	0.580	0.426	$1473.50
Test 02	180	3585	2092	287	0.613	0.385	$1436.50
Training 03	550	14662	7281	35	0.675	0.940	$41070.00
Validation 03	53	3795	2219	77	0.626	0.408	$772.50
Test 03	53	3625	2393	73	0.599	0.421	$472.00
Training 04	2166	13892	6237	233	0.713	0.903	$210603.00
Validation 04	287	3301	2159	397	0.584	0.420	$4348.00
Test 04	274	3584	1793	493	0.628	0.357	$874.00
ITERATION 2							
Training 01	1191	15559	5605	173	0.744	0.873	$61870.50
Validation 01	145	3844	1958	197	0.649	0.424	$749.00
Test 01	83	3516	2447	98	0.586	0.459	$2557.00
Training 02	1600	15420	5491	17	0.756	0.989	$102211.50
Validation 02	55	3553	2475	61	0.587	0.474	$4014.50
Test 02	80	3759	2192	113	0.625	0.415	$596.00
Training 03	245	14635	7586	62	0.661	0.798	$16504.00
Validation 03	13	3845	2259	27	0.628	0.325	$655.00
Test 03	17	3683	2429	15	0.602	0.531	$341.50
Training 04	2193	13881	6210	244	0.714	0.900	$210464.00
Validation 04	303	3290	2143	408	0.585	0.426	$3242.00
Test 04	272	3574	1795	503	0.626	0.351	$600.00
ITERATION 3							
Training 01	190	15091	5706	641	0.718	0.630	$41319.50
Validation 01	133	3766	1970	275	0.635	0.326	$88.50
Test 01	87	3472	2443	142	0.579	0.380	$313.00
Training 02	1060	15257	6031	180	0.724	0.855	$62367.50
Validation 02	83	3499	2447	115	0.583	0.419	$2131.00
Test 02	111	3689	2161	183	0.618	0.378	$668.00
Training 03	1152	14530	6679	167	0.696	0.873	$75680.50
Validation 03	130	3663	2142	209	0.617	0.383	$420.50
Test 03	121	3495	2325	203	0.589	0.373	$540.50
Training 04	639	14106	7764	19	0.655	0.971	$52713.50
Validation 04	57	3631	2389	67	0.600	0.460	$2903.00
Test 04	52	3969	2015	108	0.654	0.325	$370.00

All models showed positive PNLs in all datasets with minor differences between the validation- and test sets on average. The differences between the validation- and test sets for model accuracy and precision were also insignificant. There was a small loss in accuracy between the training set and the validation set and a noticeable decay in precision. One interesting observation is that even though the number of bad trades (False Positives) outnumber the number of good trades (True Positives) in both the validation- and test sets, all models still yielded positive PNLs. This suggests that the models identified major increases in price levels correctly. Compared to the previous study [1], where the main performance measure was based on the ratio between True Positives and False Positives, the models were more aggressive when using the PNL as the performance measure. Also, negative PNLs were encountered in the previous study. This suggest that basing the performance measure on the PNL as compared to the ratio between the True Positives and False Positives is more profitable. The averaged performance measures for each iteration are shown in Table 14.4.

Table 14.4 HTM Average Performance

Dataset	TP	TN	FN	FP	Accuracy	Precision	PNL
ITERATION 1							
Training	1464	14897	6066	101	0.726	0.921	$111933.00
Validation	126	3633	2212	173	0.612	0.425	$1958.50
Test	128	3602	2201	213	0.607	0.491	$873.00
ITERATION 2							
Training	1307	14874	6223	124	0.719	0.890	$97762.50
Validation	129	3633	2209	173	0.612	0.412	$2165.00
Test	113	3633	2216	182	0.610	0.439	$1023.50
ITERATION 3							
Training	985	14746	6545	252	0.698	0.832	$58020.00
Validation	101	3640	2237	166	0.609	0.397	$1386.00
Test	93	3656	2236	159	0.610	0.364	$473.00
TOTAL (ALL ITERATIONS)							
Training	1252	14839	6278	159	0.714	0.881	$89238.50
Validation	119	3635	2219	171	0.611	0.411	$1836.50
Test	111	3630	2217	185	0.609	0.431	$790.00

14.2 ANN Optimization Results

Table 14.5 summarizes the resulting ANN parameters produced by the genetic algorithm.

Table 14.5 ANN Optimization Results

Parameter	Dataset 01	Dataset 02	Dataset 03	Dataset 04
		ITERATION 1		
Topology	[15]	[16]	[15,7]	[16]
Learning Rate	0.102	0.067	0.070	0.103
Momentum	0.067	0.005	0.020	0.007
		ITERATION 2		
Topology	[14]	[16]	[9]	[16]
Learning Rate	0.211	0.105	0.102	0.101
Momentum	0.010	0.011	0.029	0.008
		ITERATION 3		
Topology	[15,13]	[16]	[15,15]	[15]
Learning Rate	0.107	0.105	0.137	0.104
Momentum	0.048	0.019	0.005	0.008

According to the literature, most neural network solutions trained on financial time series yield better performance using one hidden layer, as opposed to multiple hidden layers. Therefore, the abundance of single-layer topologies produced by the genetic algorithm is in line with other empirical studies. The genetic algorithm also produced a slightly higher number of neurons in the hidden layer as compared to the number of inputs.

A low learning rate and momentum was selected by the genetic algorithm. In fact, the momentum was close to zero in all cases. A common approach when using backpropagation with neural networks is to start with a relatively high learning rate and momentum and gradually decrease both values as training progresses. This usually increases convergence and makes the neural network concentrate on a more local search as the values decay. Although no decay was used in this implementation, the selected learning rate and momentum values agree with the theory in respect to yielding small final values.

Table 14.6 shows the performance results for the fittest ANN individuals in the final generation for all three iterations. Negative PNLs are shown within parentheses.

Table 14.6 ANN Performance

Dataset	TP	TN	FN	FP	Accuracy	Precision	PNL
ITERATION 1							
Training 01	1	15730	6800	2	0.698	0.333	$41.00
Validation 01	1	4631	2390	0	0.660	1.000	$184.50
Test 01	1	4048	2986	0	0.576	1.000	$147.00
Training 02	76	15322	7015	121	0.683	0.386	$334.00
Validation 02	24	4025	2963	23	0.576	0.511	$1384.00
Test 02	30	4426	2550	29	0.633	0.508	$1023.00
Training 03	39	14652	7799	45	0.652	0.464	$1235.50
Validation 03	11	4448	2569	7	0.634	0.611	$471.00
Test 03	9	4201	2810	16	0.598	0.360	$450.00
Training 04	685	13150	7718	983	0.614	0.411	$608.50
Validation 04	165	3991	2654	226	0.591	0.422	$3989.50
Test 04	101	4506	2233	185	0.656	0.353	$129.50
ITERATION 2							
Training 01	1	15730	6800	2	0.698	0.333	$28.50
Validation 01	2	4631	2389	0	0.660	1.000	$156.50
Test 01	9	4040	2978	8	0.576	0.529	($1.00)
Training 02	15	15426	7076	17	0.685	0.469	$154.00
Validation 02	9	4042	2978	6	0.576	0.600	$1280.00
Test 02	3	4453	2577	2	0.633	0.600	$235.00
Training 03	26	14664	7812	33	0.652	0.441	$748.00
Validation 03	10	4446	2570	9	0.633	0.526	$355.50
Test 03	6	4212	2813	5	0.599	0.545	$192.00
Training 04	138	13922	8265	211	0.624	0.395	$1065.50
Validation 04	33	4178	2786	39	0.589	0.458	$1696.50
Test 04	28	4636	2306	55	0.664	0.337	$113.50
ITERATION 3							
Training 01	2	15728	6799	4	0.698	0.333	$7.00
Validation 01	2	4628	2389	3	0.659	0.400	$85.00
Test 01	3	4045	2984	3	0.575	0.500	$219.50
Training 02	21	15417	7070	26	0.685	0.447	$121.50
Validation 02	10	4040	2977	8	0.576	0.556	$1271.00
Test 02	7	4445	2573	10	0.633	0.412	($126.00)
Training 03	20	14676	7818	21	0.652	0.488	$589.50
Validation 03	6	4451	2574	4	0.634	0.600	$332.50
Test 03	5	4211	2814	6	0.599	0.455	$17.00
Training 04	320	13754	8083	379	0.625	0.458	$15978.00
Validation 04	58	4143	2761	74	0.597	0.439	$1404.00
Test 04	14	4667	2320	24	0.666	0.368	$448.50

The PNL was positive in all datasets except two test sets. Furthermore, the difference in model accuracy and precision between the training-, validation- and test sets was insignificant. Even though many ANN models had more bad trades (False Positives) than good trades (True Positives), the PNL was positive for all models but two when applied to the test sets. As with the HTM models, the ANN models identified major increases in price levels correctly. The averaged performance measures for each iteration are shown in Table 14.7. As can be seen, all averaged PNLs are positive and model accuracy and precision are similar to each other in the training-, validation- and test sets.

Table 14.7 ANN Average Performance

Dataset	TP	TN	FN	FP	Accuracy	Precision	PNL
ITERATION 1							
Training	200	14714	7333	288	0.662	0.399	$555.00
Validation	50	4274	2644	64	0.615	0.636	$1507.00
Test	35	4295	2645	57	0.616	0.555	$437.50
ITERATION 2							
Training	45	14936	7488	66	0.665	0.410	$499.00
Validation	14	4324	2681	13	0.617	0.646	$872.00
Test	12	4335	2669	17	0.618	0.503	$135.00
ITERATION 3							
Training	91	14894	7442	107	0.665	0.432	$4174.00
Validation	19	4316	2675	22	0.617	0.499	$773.00
Test	7	4342	2673	11	0.618	0.434	$140.00
TOTAL (ALL ITERATIONS)							
Training	112	14848	7421	154	0.664	0.413	$1742.50
Validation	28	4305	2667	33	0.616	0.594	$1051.00
Test	18	4324	2662	29	0.617	0.497	$237.50

14.3 HTM Versus ANN Performance Comparison

The results show that the HTM models outperformed the neural network models with regards to profits made, yielding 2-8 times as much in PNL as the neural networks. Both technologies had above average accuracy in all datasets, whereas the precision was slightly below average in the validation- and test sets. Overall, both models were profitable. The positive PNLs in both the validation- and test sets suggest that both technologies produced models with good generalization abilities.

All datasets had considerably more samples belonging to class 1 than class 0. In fact, almost twice as many samples belonged to class 1 as compared to class 0 in all datasets. Thus, under these circumstances, the models did quite well.

15 Discussion

This chapter discusses the results and compares them with the results in the previous study [1]. The results are also compared to the theory.

15.1 Comparison to theory and hypotheses

Chapter 11 elaborated on theoretical aspects from chapters 3 through 10 which resulted in a number of hypotheses, used to guide the experimental design. The efficient market hypothesis (EMH) states that, in theory, financial markets are efficient to such a degree that it is impossible to consistently outperform the market. The EMH also claims that there is no serial dependencies between prices, therefore the direction of financial markets cannot be predicted by projecting historical price patterns into the future. As discussed in chapter 3, the trading community has refuted the EMH through empirical results, hence proving that financial markets, in practice, are not as efficient as suggested by the EMH. As a result of this hypotheses (H.1a) and (H1.b) were conjectured in order to verify if an HTM model could be design to exploit market inefficiencies to yield a positive average return. The two hypotheses are reiterated below for convenience:

Using (H.1a) it was hypothesized: "For some HTM model and some validation set, if the HTM model yields a positive average return on the dataset, then the HTM model is capable of exploiting financial market inefficiencies".

The results clearly show (Table 12.3) that there exists an HTM model that yields a positive average return on some validation dataset, hence the hypothesis is accepted and the EMH rejected. Although, a further elaboration in chapter 6 stated that this is not enough to verify that HTM models are capable of exploiting financial market inefficiencies *consistently*. In order to verify consistency, multiple iterations over the datasets, as described by the theory in chapter 6, were necessary. Hypothesis (H1.b) makes this claim.

Using (H.1b) it was hypothesized: "For some HTM model and some validation set over all iterations, if the HTM model yields a positive average return on the dataset, then the HTM model is capable of consistently exploiting financial market inefficiencies".

The results (Table 12.3) once again show that the hypothesis is accepted since the HTM model yielded positive average returns on all validation sets in all three iterations.

The next theoretical aspect concerned itself with optimization. Chapter 8 presents the HTM technology and details a number of tunable model parameters. Furthermore, the genetic algorithm (GA) was described in chapter 9, where it was stated that the GA is able to explore and exploit a solution space consisting of a population of individuals, representing the solution to a problem. Hypothesis (H.2a) therefore stated that it was possible to optimize HTM models using the GA, hence yielding positive average returns on all validation datasets.

Hypothesis (H2.a) was defined as: "For some GA-optimized HTM model and all validation sets, if the HTM model yields a positive average return on the datasets, then the HTM model can be optimized for financial markets using the GA algorithm".

The results from the experiments given in Table 12.3 show that the HTM models produced positive average returns on all validation datasets, hence implying that the GA can be used to optimize HTM models. The discussion in chapter 11 then identified a problem with this hypothesis due to the stochastic nature of the GA, i.e. if the same input data is fed to the GA, it will most likely not produce the exact same result twice. Therefore, a second hypothesis (H2.b) was defined, stating that the stochastic GA algorithm optimizes HTM models every time it is employed.

Hypothesis (H2.b) stated that: "For some GA-optimized HTM model and all validation sets in all iterations, if the HTM model yields a positive average return on the datasets, then the HTM model can consistently be optimized for financial markets using the GA algorithm".

Once again, by referring to Table 12.3, it is obvious that the hypothesis holds, since the HTM model had a positive average return on all validation datasets in all iterations.

The third hypothesis dealt with HTM models' generalization abilities. The theory was once again consulted in order to define the hypothesis. Theory in chapter 6 states that a model that performs well, not only on a validation dataset, but also on a test dataset, consisting of previously unseen data, shows good generalization abilities. This was conjectured by hypothesis (H.3).

Hypothesis (H.3a) states: "For some HTM model and all validation- and test datasets, if the HTM model yields a positive average return on the datasets, then the HTM model generalizes well to novel data".

The hypothesis is supported by the results shown in Table 12.3, where the HTM model yielded positive average returns in all validation- and test sets. Hypothesis (H.3b) further guaranteed consistent generalization ability.

Using hypothesis (H.3b) it was hypothesized: "For some HTM model and all validation- and test datasets in all iterations, if the HTM model yields a positive average return on the datasets, then the HTM model generalizes well to novel data".

According to Table 12.3, the HTM model showed positive average returns in all validation- and test datasets over all iterations, hence accepting the hypothesis.

Finally, with support of chapters 7 and 10, artificial neural networks (ANNs) were announced as being the benchmark technology with respect to predictive models. In order to assess the performance of the HTM technology against the benchmark technology, recurrent neural networks were therefore GA-optimized and trained, validated and tested

on the same datasets. The fourth and final hypothesis stated that the HTM technology outperforms the ANN technology according to hypothesis (H.4).

Hypothesis (H.4) states that: "For some GA-optimized HTM model and some GA-optimized ANN model, on all test datasets in all iterations, if the HTM model yields a higher positive average return on the datasets than the ANN model, then the HTM model consistently outperforms the ANN model".

By examining the tables Table 12.3, Table 12.4, Table 12.6 and Table 12.7 the hypothesis once again holds, since the HTM model yielded a higher average positive return on all test datasets as compared to the ANN model.

15.2 Comparison to previous work

The genetic algorithm produced HTM models with three-layer topologies, which were also preferred in the previous study. One significant difference between this study and the previous, is that a manual search was utilized in the previous study. In this paper, the genetic algorithm was able to search for the best possible settings for most HTM parameters (the previous study limited the available HTM parameters and their values when manually searching for the best possible models). As a consequence, the genetic algorithm was able to fine-tune parameter settings for nodes in each layer individually. This proved beneficial for generating more favorable PNLs. Another significant difference between the two studies is the performance measure used to guide the search for the best performing models. In the previous study, the ratio between the true positives (good trades) and false positives (bad trades) was adopted, whereas the actual PNL (Profit-and-Loss) was used in this study. This also increased model profitability.

The max distance setting and the maximum number of coincidences and groups used in various layers showed greater variability in this paper as compared to the previous. This suggests that the genetic algorithm was able to adjust the number of spatial- and temporal patterns in each layer more appropriately.

In the previous study, network topology, max distance, coincidences and groups were the only HTM parameters considered. This paper optimized a much larger subset of HTM parameters. By letting the genetic algorithm optimize the spatial pooler's inference algorithm, better inferences with regards to novel spatial patterns were possible. As was observed in the results, the Gaussian inference algorithm was chosen for all nodes, suggesting that the K^{th} Root Product algorithm is inappropriate for financial time series, or at least for the chosen dataset. Furthermore, the genetic algorithm was able to tweak the Sigma setting for the Gaussian inference algorithm for each layer independently, which in essence let each layer adapt better to spatial noise in its inputs.

The temporal pooler's inference algorithm, with corresponding parameters, was also optimized independently per layer by the genetic algorithm. Additionally, Time-Based Inference (TBI) was used in this paper (as opposed to the previous study) which let the HTM models make better use of temporal pattern transitions over a larger time span.

Using the fee-adjusted PNL as the fitness function caused the genetic algorithm to eliminate too aggressive models. In fact, the number of positives were a fraction of the positives produced by the HTM models obtained in the previous study. One interesting observation was that even though the number of bad trades (false positives) outnumbered the number of good trades (true positives), the PNL was still positive. In the previous study, models with poor true positive / false positive ratios were discarded yielding lower PNL scores as compared to this paper. This suggest that using the fee-adjusted PNL as the performance measure is more profitable even though the number of bad trades outnumbers the number of good trades.

The neural network models produced by the genetic algorithm had one hidden layer and performed well in almost all data sets. In the previous study, the neural network had two hidden layers and performed much worse. This suggests that one hidden layer with more neurons is a better choice. The genetic algorithm was also able to tweak the networks' learning rate and momentum which further enhanced model performance. One important difference between the two studies is the fact that recurrent neural networks were used in this paper, whereas fully-connected feed-forward neural networks without time-lagged inputs were used in the previous study. This obviously has an impact on model performance, since the feed-forward neural networks were only able to model a static input space.

Comparing the two technologies, the HTM models outperformed the neural network models with regards to the PNL. Overall, both technologies were profitable.

15.3 Reliability, Replicability and Validity

As stated in chapter 2, reliability has to do with the stability of measurements used during the study. In this study it relates directly to the technical indicators derived from the financial time series. The technical indicators are highly consistent and standardized measures which therefore imply a high degree of reliability. The technical indicators are detailed in Appendix A. Furthermore, neural networks, hierarchical temporal memory and evolutionary learning methods are all stochastic in nature, and hence produce non-deterministic models. Despite this, the reliability of the results are assured by using cross-validation and multiple iterations. This produces an average measurement and reduces variation.

Replicability is concerned with the reproducibility of the experiments. The method used during data acquisition, preprocessing, feature extraction, modeling and analysis is meticulously described in chapters 12 and 13, hence yielding a high degree of replicability.

Measurement validity relates to if a derived measure really reflects the concept it is supposedly measuring. As with reliability, the same argument can be used to suggest a high degree of measurement validity, i.e. since the technical indicators, used as the derived measures of characteristics of financial markets, are standardized measures, they also imply a high degree of measurement validity. Furthermore, the measures used to

assess model performance are based on well-known statistical measures and therefore also ensure a high degree of measurement validity.

The internal validity is concerned with the causality between the independent variables and the dependent variable, i.e. if the independent variables really account for the variation in the dependent variable. No assessment was conducted in this study with regards to the correlation between each independent variable and the dependent variable. The only assessment of the internal validity of the results incorporates the collective casual effect of the independent variables on the dependent variable, which proved to yield profitable results.

The external validity is a direct assessment of the generalization ability of the study. The models' generalization abilities were ensured through cross-validation. In a bigger picture, it is hypothesized that the method will generalize to other financial markets, whereas it is doubtful if the results will, since the discovered patterns in the E-mini S&P 500 market might differ from other markets (such as fixed income, forex, commodities).

16 Conclusions and Future Work

This chapter contains conclusive remarks and makes references to the main research objectives as outlined in Chapter 1. The chapter also elaborates on suggested future work.

16.1 Conclusions

The results show that the genetic algorithm succeeded in finding predictive models with good performance and generalization ability. Although the HTM models outperformed the neural networks with regards to PNL, both technologies yielded profitable results with above average accuracy. This paragraph essentially summarizes the fulfillment of all four research objectives as defined in subchapter 1.4.

Research objective (RO.1) was to implement profitable, predictive HTM-based classification models for financial time series. This included selecting and partitioning a financial time-series, extracting a dozen technical indicator-based features, defining a classification task, defining a trading strategy and configuring, training, validating and testing predictive HTM classification models. The results obtained in subchapter 14.1 show that the implemented predictive HTM classifiers were profitable in all financial datasets, which accepts hypotheses (H.1a) and (H.1b), fulfilling research objective (RO.1) and hence, answering research question (RQ.1).

The optimization results show that the HTM models prefer a three-layer network topology with a variable max distance setting and number of coincidences and groups in each layer when applied to a financial time series based on the E-mini S&P 500 stock index future market, with a minute time resolution. Furthermore, a Gaussian inference algorithm with a Sigma setting close to one standard deviation was preferred by the spatial pooler in all nodes throughout the network hierarchy. With respect to the temporal pooler's inference algorithm, flash inference and time-based inference were equally prevalent amongst the models' first layer (the only permissible layer for time-based inference). The settings for the associated parameters (transition memory for flash

inference versus sequencer model complexity, window length and window count for time-based inference) differed throughout the hierarchy, suggesting that individual settings were preferred for each node. Similarly, the neural network preferred a single hidden layer with a node count close to the dimensionality of its input and small values for its learning rate and momentum. These results, together with the performance results obtained in subchapter 14.1, support hypotheses (H.2a) and (H.2b) and correspond to research objective (RO.2) and research question (RQ.2), i.e. optimize and evaluate model performance using evolutionary learning. This also suggests that the chromosome mappings for the genetic algorithm and the chosen fitness function were successful.

The modified cross-validation technique employed in this paper showed insignificant differences between the validation sets and the test sets with regard to both PNL and model accuracy, hence suggesting a favorable model generalization ability. This result supports hypotheses (H.3a) and H.3b), fulfilling research objective (RO.3) and hence, answering research question (RQ.3); evaluate model generalization ability using a modified cross-validation technique.

Lastly, research objective (RO.4), benchmark HTM technology performance against recurrent neural networks, is covered by the comparison made between the results obtained for the HTM models and the neural network models. The results suggest that the HTM technology outperforms, or is at least as good as, the neural network technology with regards to the PNL performance measure, hence supporting hypothesis (H.4) and answering research question (RQ.4).

16.2 Future Work

Using a fee-adjusted PNL as the fitness function, caused the genetic algorithm to eliminate too aggressive models. This yields risk-averse models, but also reduces their trading tendency. More aggressive models were obtained in a previous study [1] where the rate between true and false positives was used as a performance measure. This raises the question if a balance between aggressive and risk-averse trading models can be achieved by combining the two performance measures in a future endeavor.

Feature vectors of normalized technical indicators proved to contain enough information to train profitable predictive models of financial markets. This paper only used 12 technical indicators. There are a plentitude of other technical and fundamental indicators to choose from. Investigating more technical and fundamental indicators is therefore appropriate for future work. One important preprocessing step, not investigated in this paper, is the various feature subset selection techniques available to find feature combinations with good predictive power. In a future attempt, considerably more focus should be given to this important step.

This study only benchmarked the HTM technology against neural networks. It would be interesting to incorporate more machine learning technologies into the study. Furthermore, the financial time series used in this study was based on a stock index derivative, with a minute time resolution, using a ten-minute time horizon. Future

research could investigate the behavior of the predictive models on different asset classes, using different time resolutions and time horizons.

Finally, a buy-and-hold trading strategy was used in this paper. A future attempt would be to investigate the profitability of using different trading strategies.

References

[1] P. Gabrielsson, R. König, and U. Johansson, "Hierarchical Temporal Memory-Based Algorithmic Trading of Financial Markets," in *2012 IEEE Symposium on Computational Intelligence for Financial Engineering and Economics (CIFEr)*, 2012.

[2] "Informatics," *Wikipedia*, 2012. [Online]. Available: http://en.wikipedia.org/wiki/Informatics_(academic_field). [Accessed: 04-Feb-2012].

[3] R. Das, J. E. Hanson, J. O. Kephart, and G. Tesauro, "Agent-Human Interactions in the Continuous Double Auction," in *Proceedings of the 17th International Joint Conference on Artificial Intelligence*, 2001, pp. 1169–1176.

[4] M. de Luca and D. Cliff, "Human-Agent Auction Interactions: Adaptive-Aggressive Agents Dominate," in *Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-2011)*, 2011.

[5] E. Fama, "Efficient Capital Markets: A Review of Theory and Empirical Work," *The Journal of Finance*, vol. 25, no. 2, pp. 383–417, 1970.

[6] A. Bryman and E. Bell, *Business Research Methods*. OUP Oxford, 2011.

[7] V. Smith, "An Experimental Study of Competitive Market Behavior," *Journal of Political Economy*, vol. 70, no. 2, pp. 111–137, 1962.

[8] E. Fama, "The Behavior of Stock-Market Prices," *The Journal of Business*, vol. 38, no. 1, pp. 34–105, 1965.

[9] J. Schwager, *Futures: Fundamental Analysis*. Wiley, 1995.

[10] W. Hamilton, *The Stock Market Barometer*. Wiley, 1998.

[11] R. Rhea, *The Dow Theory*. Fraser Publishing, 1994.

[12] J. Murphy, *Technical Analysis of the Financial Markets*. New York Institute of Finance, 1999.

[13] B. Malkiel, *A Random Walk Down Wall Street*. W. W. Norton & Company, 2011.

[14] M. Kendall, "The Analysis of Economic Time Series - Part I: Prices," *Journal of the Royal Statistical Society*, vol. 116, no. 1, pp. 11–34, 1953.

[15] U. Fayyad, G. Piatetsky-shapiro, and P. Smyth, "From Data Mining to Knowledge Discovery in Databases," *AI Magazine*, vol. 17, pp. 37–54, 1996.

[16] "Crisp-dm 1.0," 2000. [Online]. Available: ftp://ftp.software.ibm.com/software/analytics/spss/support/Modeler/Documentation/14/UserManual/CRISP-DM.pdf. [Accessed: 15-Feb-2012].

[17] S. S. Stevens, "On the Theory of Scales of Measurement," *Science (New York, N.Y.)*, vol. 103, no. 2684, pp. 677–80, Jun. 1946.

[18] Fisher, "The Iris Dataset," *UCI Machine Learning Repository*, 1936. [Online]. Available: http://archive.ics.uci.edu/ml/datasets/Iris. [Accessed: 17-Feb-2012].

[19] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE : Synthetic Minority Over-sampling Technique," *Artificial Intelligence*, vol. 16, no. 1, pp. 321–357, 2002.

[20] P.-N. Tan, M. Steinbach, and V. Kumar, *Introduction to Data Mining*. Addison Wesley, 2009.

[21] L. Breiman, "Bagging Predictors," in *Machine Learning*, 1996, vol. 24, pp. 123–140.

[22] R. E. Schapire, "The strength of weak learnability," *Machine Learning*, vol. 5, no. 2, pp. 197–227, Jun. 1990.

[23] L. Sherwood, *Human Physiology: From Cells to Systems*. Brooks Cole, 2009.

[24] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The Bulletin of Mathematical Biophysics*, vol. 5, no. 4, pp. 115–133, Dec. 1943.

[25] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological Review*, vol. 65, no. 6, pp. 386–408, 1958.

[26] M. Minsky and S. Papert, *Perceptrons: An Introduction to Computational Geometry*, 1st ed. MIT Press, 1969, p. 268.

[27] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, Oct. 1986.

[28] J. L. McClelland and D. E. Rumelhart, *Parallel Distributed Processing*, 1st ed. A Bradford Book, 1987, p. 576.

[29] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural Networks*, vol. 6, no. 4, pp. 525–533, Jan. 1993.

[30] M. Riedmiller and H. Braun, “A direct adaptive method for faster backpropagation learning: the RPROP algorithm,” in *IEEE International Conference on Neural Networks*, 1993, pp. 586–591.

[31] I. Arel, D. C. Rose, and T. P. Karnowski, “Deep Machine Learning - A New Frontier in Artificial Intelligence Research [Research Frontier],” *IEEE Computational Intelligence Magazine*, vol. 5, no. 4, pp. 13–18, Nov. 2010.

[32] V. Mountcastle, “An Organizing Principle for Cerebral Function: The Unit Model and the Distributed System,” in *The Mindful Brain*, V. Mountcastle and G. Edelman, Eds. Cambridge, MA, USA: MIT Press, 1978, pp. 7–50.

[33] J. Hawkins and S. Blakeslee, *On Intelligence*. St. Martin’s Griffin, 2004.

[34] Y. Bengio, “Learning Deep Architectures for AI,” *Foundations and Trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.

[35] V. Mountcastle, “Modality and topographic properties of single neurons of cat’s somatic sensory cortex,” *Journal of neurophysiology*, vol. 20, no. 4, pp. 408–434, 1957.

[36] D. George and J. Hawkins, “A hierarchical Bayesian model of invariant pattern recognition in the visual cortex,” in *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, 2005, vol. 3, pp. 1812–1817.

[37] J. Pearl, *Probabilistic reasoning in intelligent systems: networks of plausible inference*, Edition 1. Morgan Kaufmann, 1988, p. 552.

[38] D. George, “How The Brain Might Work: A Hierarchical And Temporal Model For Learning And Recognition,” Stanford University, 2008.

[39] D. George, “How to make computers that work like the brain,” in *Proceedings of the 46th ACM/IEEE Design Automation Conference (DAC)*, 2009, pp. 420–423.

[40] “NuPIC 1.7.1,” 2011. [Online]. Available: http://www.numenta.com/legacysoftware.php. [Accessed: 15-Jan-2012].

[41] J. Hawkins and D. George, “Hierarchical Temporal Memory - Concepts, Theory and Terminology,” 2006. [Online]. Available: http://www.numenta.com/htm-overview/education/Numenta_HTM_Concepts.pdf. [Accessed: 15-Jan-2012].

[42] D. George and B. Jaros, “The HTM Learning Algorithms,” 2007. [Online]. Available: http://www.numenta.com/htm-overview/education/Numenta_HTM_Learning_Algos.pdf. [Accessed: 15-Jan-2012].

[43] C. Darwin, *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life*, 1st ed. John Murray, 1859, p. 502.

[44] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, 1st ed. A Bradford Book, 1975, p. 211.

[45] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, 1st ed. A Bradford Book, 1992, p. 840.

[46] C.-F. Tsai and Y.-C. Hsiao, "Combining multiple feature selection methods for stock prediction: Union, intersection, and multi-intersection approaches," *Decision Support Systems*, vol. 50, no. 1, pp. 258–269, Dec. 2010.

[47] S. Thawornwong and D. Enke, "The adaptive selection of financial and economic variables for use with artificial neural networks," *Neurocomputing*, vol. 56, pp. 205–232, Jan. 2004.

[48] D. Enke and S. Thawornwong, "The use of data mining and neural networks for forecasting stock market returns," *Expert Systems with Applications*, vol. 29, no. 4, pp. 927–940, Nov. 2005.

[49] C.-J. Lu, "Integrating independent component analysis-based denoising scheme with neural network for stock price prediction," *Expert Systems with Applications*, vol. 37, no. 10, pp. 7056–7064, Oct. 2010.

[50] J.-Z. Wang, J.-J. Wang, Z.-G. Zhang, and S.-P. Guo, "Forecasting stock indices with back propagation neural network," *Expert Systems with Applications*, vol. 38, no. 11, pp. 14346–14355, May 2011.

[51] E. L. de Faria, M. P. Albuquerque, J. L. Gonzalez, J. T. P. Cavalcante, and M. P. Albuquerque, "Predicting the Brazilian stock market through neural networks and adaptive exponential smoothing methods," *Expert Systems with Applications*, vol. 36, no. 10, pp. 12506–12509, Dec. 2009.

[52] T. Chavarnakul and D. Enke, "Intelligent technical analysis based equivolume charting for stock trading using neural networks," *Expert Systems with Applications*, vol. 34, no. 2, pp. 1004–1017, Feb. 2008.

[53] 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*, vol. 7, no. 2, pp. 569–576, Mar. 2007.

[54] J. D. Schaffer, D. Whitley, and L. J. Eshelman, "Combinations of genetic algorithms and neural networks: a survey of the state of the art," in *[Proceedings]*

COGANN-92: International Workshop on Combinations of Genetic Algorithms and Neural Networks, pp. 1–37.

[55] K. S. Tang, "Genetic structure for NN topology and weights optimization," in *1st International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications (GALESIA)*, 1995, vol. 1995, no. 414, pp. 250–255.

[56] C. Bergmeir and J. M. Benítez, "On the use of cross-validation for time series predictor evaluation," *Information Sciences*, vol. 191, no. 15 May 2012, pp. 192–213, May 2012.

[57] C.-F. Tsai, Y.-C. Lin, D. C. Yen, and Y.-M. Chen, "Predicting stock returns by classifier ensembles," *Applied Soft Computing*, vol. 11, no. 2, pp. 2452–2459, Mar. 2011.

[58] G. Zhang, B. Eddy Patuwo, and M. Y. Hu, "Forecasting with artificial neural networks:: The state of the art," *International Journal of Forecasting*, vol. 14, no. 1, pp. 35–62, 1998.

[59] G. S. Atsalakis and K. P. Valavanis, "Surveying stock market forecasting techniques – Part II: Soft computing methods," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5932–5941, Apr. 2009.

[60] B. Widrow and M. A. Lehr, "30 Years of Adaptive Neural Networks: Perceptron, Madaline and Backpropagation," in *Proceedings of the IEEE, 78 (9)*, 1990, pp. 1415–1442.

[61] J. Thornton, T. Gustafsson, M. Blumenstein, and T. Hine, "Robust Character Recognition using a Hierarchical Bayesian Network," *AI 2006: Advances in Artificial Intelligence*, vol. 4304, pp. 1259–1264, 2006.

[62] J. V. Doremalen and L. Boves, "Spoken Digit Recognition using a Hierarchical Temporal Memory," in *INTERSPEECH 2008 - 9th Annual Conference of the International Speech Communication Association (ISCA)*, 2008, pp. 2566–2569.

[63] J. Thornton, J. Faichney, M. Blumenstein, and T. Hine, "Character Recognition Using Hierarchical Vector Quantization and Temporal Pooling," in *AI 2008 Proceedings of the 21st Australasian Joint Conference on Artificial Intelligence: Advances in Artificial Intelligence*, vol. 5360, W. Wobcke and M. Zhang, Eds. Springer Berlin / Heidelberg, 2008, pp. 562–572.

[64] S. Štolc and I. Bajla, "On the Optimum Architecture of the Biologically Inspired Hierarchical Temporal Memory Model Applied to the Hand-Written Digit Recognition," *Measurement Science Review*, vol. 10, no. 2, pp. 28–49, Jan. 2010.

[65] J. Maxwell, P. Pasquier, and A. Eigenfeldt, “Hierarchical Sequential Memory for Music: A Cognitive Model,” in *Proceedings of the 10th International Society for Music Information Retrieval (ISMIR)*, 2009, pp. 429–434.

[66] D. Rozado, F. B. Rodriguez, and P. Varona, “Optimizing Hierarchical Temporal Memory for Multivariable Time Series,” in *Artificial Neural Networks – ICANN 2010*, vol. 6353, K. Diamantaras, W. Duch, and L. Iliadis, Eds. Springer Berlin / Heidelberg, 2010, pp. 506–518.

[67] D. Rozado, F. B. Rodriguez, and P. Varona, “Extending the bioinspired hierarchical temporal memory paradigm for sign language recognition,” *Neurocomputing*, vol. 79, no. 0, pp. 75–86, 2012.

[68] F. Åslin, “Evaluation of Hierarchical Temporal Memory in algorithmic trading,” University of Linköping, 2010.

[69] S. Marsland, *Machine Learning: An Algorithmic Perspective*, 1st ed. Chapman and Hall, 2009.

[70] StockCharts, “StockCharts.” StockCharts, 2011.

Appendix A

This appendix contains descriptions, plots and mathematical formulas for the various technical indicators used in this paper [12], [70]. The simple moving average (SMA) and exponential moving average (EMA) are not used as technical indicators in this paper, but are included since some of the other indicators are based on them. The description for the SMA below, explains some common notation used for each indicator.

Simple Moving Average (SMA)

A simple moving average is a trend-following indicator with a lag. It uses a window of size N to average prices, where N is called the indicator's period, and is also its lag. The resolution of the period depends on what is averaged, e.g. if hourly prices are averaged, the period has an hourly resolution and if daily prices are averaged, a daily resolution is obtained. At each time step, the window is shifted forward one step, thereby including the current price and discarding the oldest price. A simple moving average is calculated using closing prices for the period according to Equation A.1, where N is the window length, $p_{close}(t)$ is the current closing price and $p_{close}(t - i)$ is the closing price i time steps ago. The expression $SMA_N(t)$ means: calculate the N-period simple moving average at time t.

$$SMA_N(t) = \frac{1}{N}\sum_{i=0}^{N-1} p_{close}(t-i) \qquad \text{(A.1)}$$

Figure A.1 shows a plot containing the closing price (blue) together with its 20-period SMA (red) and 50-period SMA (green). In price charts, it is common to use a notation where the period of an indicator is given within parentheses. For example, the 20-period SMA is show as SMA(20) on a price chart, which is not to be confused with the $SMA_N(t)$ notation in the above equation where t is the time and the subscript N is the period. Some indicators use multiple periods in their calculations, e.g. the PPO indicator described below. In such cases, each period is included in the notation for the indicator on the prices chart, e.g. PPO(12,26,9).

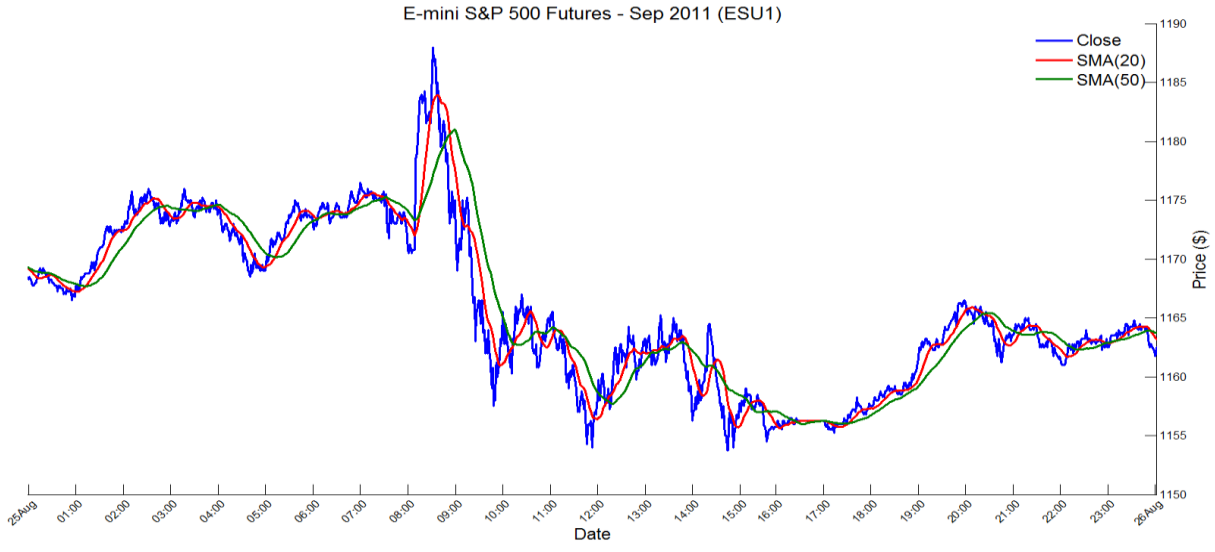

Figure A.1 Closing price with two SMAs for the E-mini S&P 500 September 2011 contract (25th of August)

Exponential Moving Average (EMA)

An exponential moving average is a trend-following indicator and is similar to a simple moving average, but reduces the lag by applying more weight to more recent prices. Therefore, exponential moving averages react quicker to turns in the trend compared to simple moving averages. The EMA is calculated according to Equation A.2, where N is the period length, $2/(N+1)$ is the weight factor, $p_{close}(t)$ is the current closing price and $EMA_N(t-1)$ is the previous EMA value (calculated in the previous time step, i.e. at time $t-1$).

$$EMA_N(t) = \frac{2}{N+1}[p_{close}(t) - EMA_N(t-1)] + EMA_N(t-1) \qquad \text{(A.2)}$$

The initial EMA calculation does not have a previous EMA value, hence the expression $EMA_N(t-1)$ is replaced with the SMA, i.e. $SMA_N(0)$, during the initial calculation. The initial EMA value is therefore calculated using Equation A.3.

$$EMA_N(0) = \frac{2}{N+1}[p_{close}(0) - SMA_N(0)] + SMA_N(0) \qquad \text{(A.3)}$$

Figure A.2 shows a plot containing the closing price (blue) together with its 20-period EMA (red) and 50-period EMA (green). By comparing the EMAs in Figure A.2 with the SMAs in Figure A.1, notice that the two EMAs turn quicker than the two SMAs. Also notice that a moving average lies on top of the price action (closing price) during a downtrend, underneath the price action during an uptrend and in the middle of the price action during a horizontal market.

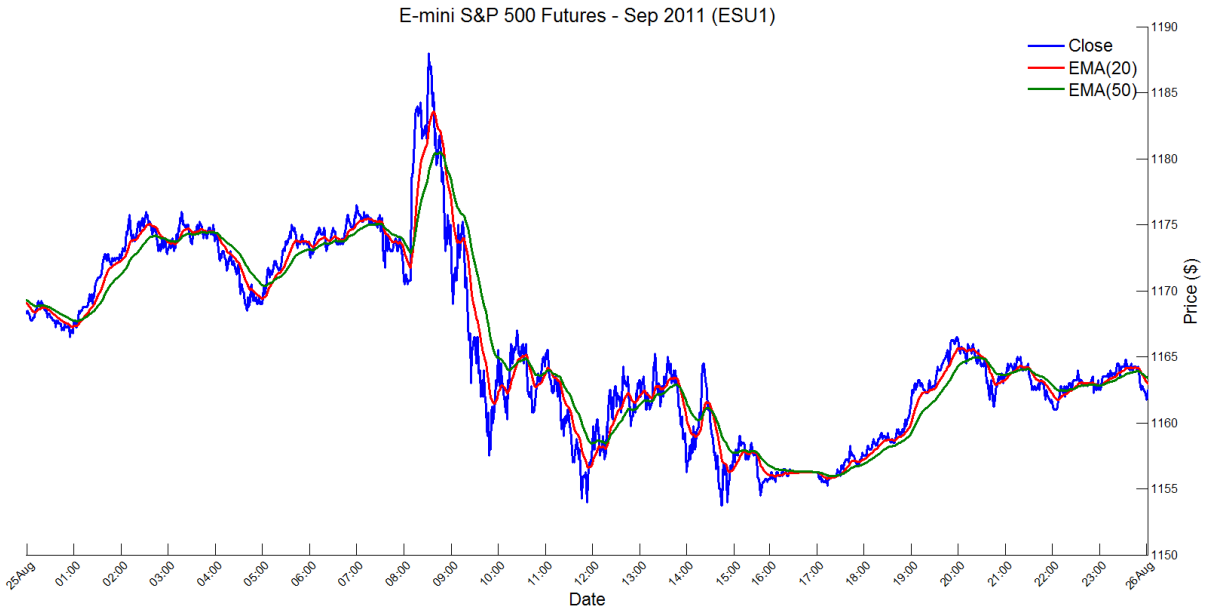

Figure A.2 Closing price with two EMAs for the E-mini S&P 500 September 2011 contract (25th of August)

Percentage Price Oscillator (PPO)

The price percentage oscillator (PPO) is a normalized version of the popular moving average convergence divergence (MACD) indicator. Both indicators are calculated by subtracting a long term exponential moving average (EMA) from a short term EMA. Additionally, the PPO indicator divides the difference between the two EMAs with the long term EMA and multiplies the result by 100. The PPO calculation is shown in Equation A.4, where $EMA_{NS}(t)$ is the short term EMA and $EMA_{NL}(t)$ is the long term EMA at time t. If the last term is omitted, the MACD indicator is obtained instead.

$$PPO_{N_S N_L}(t) = \left[EMA_{N_S}(t) - EMA_{N_L}(t)\right] \times \frac{100}{EMA_{N_L}(t)} \qquad \text{(A.4)}$$

Taking the difference between the two EMAs creates an oscillating signal and therefore the PPO and the MACD are known as momentum oscillators. The indicators oscillate around a zero line as the two EMAs cross each other. Zero line crossings and divergences are used as buy and sell signals.

The blue plot in Figure A.3 shows a PPO indicator using a 12-minute period for its short term EMA and a 26-minute period for its long term EMA, shown as PPO(12,26) in the legend.

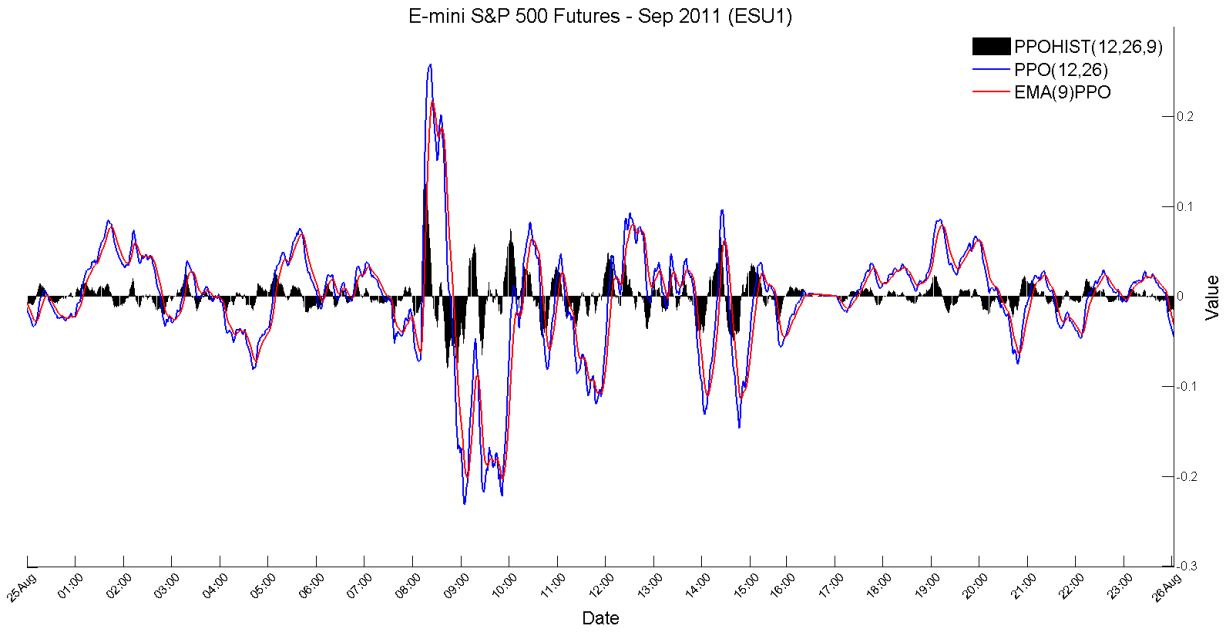

Figure A.3 PPO, its signal line and histogram for the E-mini S&P 500 September 2011 contract (25th of August)

PPO Signal Line (PPO_{EMA})

The PPO signal line, normally calculated as the 9-period EMA of the price percentage oscillator (PPO), is used to identify turns in the PPO indicator. If the PPO indicator is above its signal line, it identifies a bullish market. Conversely, the PPO indicator is below its signal line, it identifies a bearish market. The PPO signal line is calculated using Equation A.5, where $PPO_{NSNL}(t)$ is the value of the PPO indicator are time t, PPO_{EMAN} is the signal line value at time $t-1$ (previous value) and $2/(N+1)$ is the EMA weight factor. The equation is identical to Equation A.2, only here the PPO is used instead of the closing price.

$$PPO_{EMA_N}(t) = \frac{2}{N+1}\left[PPO_{N_sN_L}(t) - PPO_{EMA_N}(t-1)\right] + PPO_{EMA_N}(t-1) \quad \text{(A.5)}$$

The PPO signal line is plotted in Figure A.3 (red line) and is shown as EMA(9)PPO in the legend. The plot shows that is acts as a short term EMA of the PPO indicator.

PPO Histogram (PPO_{HIST})

The PPO-Histogram is simply calculated as the difference between the PPO indicator and its signal line. This is shown in Equation A.6, where PPO_{NSNL} is the value of the PPO indicator at time t and PPO_{EMAN} is the value of its signal line at time t.

$$PPO_{HIST_{N_sN_LN_E}}(t) = PPO_{N_sN_L}(t) - PPO_{EMA_N}(t) \quad \text{(A.6)}$$

The histogram can be positive or negative, and hence has a zero line. When the PPO indicator is above its signal line, the histogram is positive, indicating a bullish market. Conversely, a negative histogram means the PPO indicator is below its signal line, indicating a bearish market. A zero crossing occurs when the PPO indicator crosses the signal line and as the two curves diverge, the area of the histogram increases.

Figure A.3 shows a PPO histogram created from a PPO indicator with a 12-minute short term period and a 26-minute long term period, and its signal line (9-minute period EMA of the PPO indicator). The histogram is displayed as a black patch with PPOHIST(12,26,9) as its legend entry.

Relative Strength Index (RSI)

The relative strength indicator (RSI) is a momentum oscillator, designed to measure the direction and velocity of price movements. RSI is a normalized indicator that operates in the range [0,100] and defines two levels for identifying overbought and oversold market conditions. A value above 70 indicates an overbought market (i.e. a good time to close out long position and/or to sell), whereas a reading below 30 is considered an oversold market (i.e. a good time to close out short positions and/or to buy).

The RSI indicator uses a few steps in its calculation. First, the price difference $\Delta P(t)$, between the current price $p_{close}(t)$, and the previous price $p_{close}(t\text{-}1)$, is calculated according to Equation A.7.

$$\Delta P(t) = p_{close}(t) - p_{close}(t-1) \tag{A.7}$$

This is used to calculate the price gain $Gain(t)$, and price loss $Loss(t)$, using Equation A.8 and Equation A.9 respectively.

$$Gain(t) = \begin{cases} |\Delta P(t)|, & \Delta P(t) > 0 \\ 0, & \Delta P(t) \leq 0 \end{cases} \tag{A.8}$$

$$Loss(t) = \begin{cases} |\Delta P(t)|, & \Delta P(t) < 0 \\ 0, & \Delta P(t) \geq 0 \end{cases} \tag{A.9}$$

Next, the average gain (Equation A.10) and average loss (Equation A.11) are calculated, where $\mu_{Gain}(t\text{-}1)$ is the previous average gain, $\mu_{Gain}(t\text{-}1)$ is the previous average loss and N is the length of the period.

$$\mu_{Gain_N}(t) = \frac{\mu_{Gain_N}(t-1) \times (N-1) + Gain(t)}{N} \tag{A.10}$$

$$\mu_{Loss_N}(t) = \frac{\mu_{Loss_N}(t-1) \times (N-1) + Loss(t)}{N} \tag{A.11}$$

The relative strength between the average gain and average loss is then calculated using Equation A.12.

$$RS_N(t) = \frac{\mu_{Gain_N}(t)}{\mu_{Loss_N}(t)} \tag{A.12}$$

Finally, the indicator is normalized in order to produce a range between 0 and 100 using Equation A.13.

$$RSI_N(t) = 100 - \frac{100}{1 + RS_N(t)} \tag{A.13}$$

Since the previous average gain and average loss is unknown during the initial calculation in Equation A.10 and Equation A.11, the initial average gain and average loss is calculated using the sum of all gains and losses over the past N time steps according to Equation A.14 and Equation A.15 respectively.

$$\mu_{Gain_N}(t) = \frac{1}{N}\sum_{i=0}^{N-1} Gain(t-i) \tag{A.14}$$

$$\mu_{Loss_N}(t) = \frac{1}{N}\sum_{i=0}^{N-1} Loss(t-i) \tag{A.15}$$

A RSI indicator with a 14-minute period is shown in Figure A.4 (blue plot). The overbought level is shown as a red line and the oversold level as a green line. As an example, the first peak, to the left in the figure, reaching a value of 100 is a strong signal that the market is overbought. The trough, near the middle right in the figure, reaching a value just under 10 is a string signal that the market is oversold.

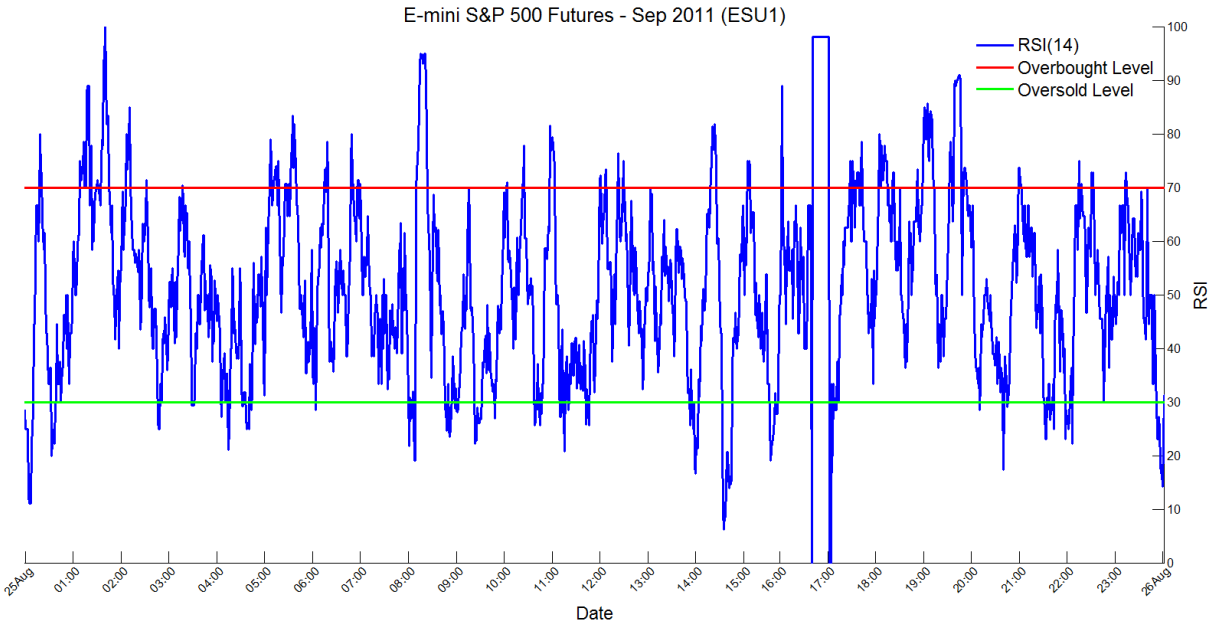

Figure A.4 RSI for the E-mini S&P 500
September 2011 contract (25th of August)

William's %R (Wm%R)

The William %R indicator is a momentum oscillator, derived from the fast stochastic oscillator (described below) indicating the current closing price level relative to the highest high price during the look-back period *N*. Just as the RSI indicator, the Wm%R indicator is normalized, but operates in the range [-100, 0] instead, and defines two levels for identifying overbought and oversold market conditions. A value above -20 indicates an overbought market, whereas a reading below -80 is considered an oversold market. The indicator is calculated using Equation A.16, where $p_{close}(t)$ is the current price, $p_{HighestHigh}(0 \leq t < N)$ is the highest high price during the look-back period N and $p_{LowestLow}(0 \leq t < N)$ is the lowest low price during the look-back period N.

$$\%R_N(t) = \frac{p_{HighestHigh}(0 \leq t < N) - p_{close}(t)}{p_{HighestHigh}(0 \leq t < N) - p_{LowestLow}(0 \leq t < N)} \times (-100) \qquad \text{(A.16)}$$

Figure A.5 show a William %R indicator (blue plot) using a 14-minute look-back period, together with its overbought level (red line) and oversold level (green line).

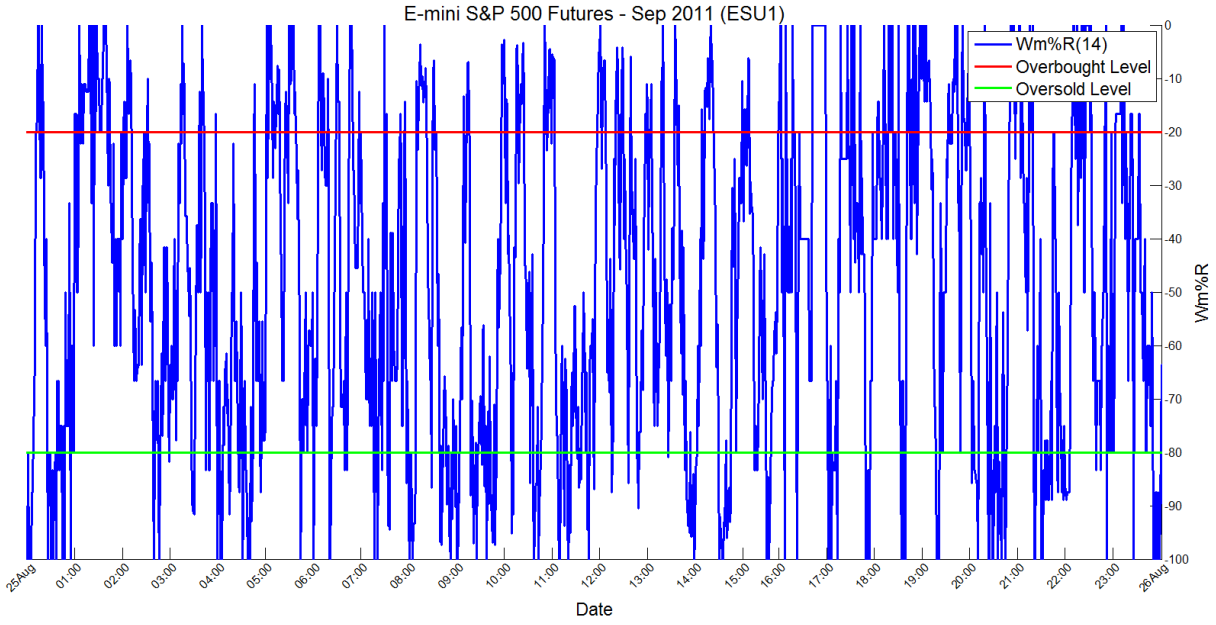

Figure A.5 Wm%R for the E-mini S&P 500 September 2011 contract (25th of August)

Normalized Volatility Indicator (NVI)

The normalized volatility indicator (NVI) measures the volatility in the market, instead of price direction, and is created by dividing the average true range (ATR) indicator with the closing price. The indicator is calculated using a number of steps and is based on the chosen period length *N*.

First, the true range (TR) is calculated from three separate calculations; $Method_1$ (Equation A.17), $Method_2$ (Equation A.18) and $Method_3$ (Equation A.19), where $p_{high}(t)$ is the high price at time *t*, $p_{low}(t)$ is the low price at time *t* and $p_{close}(t-1)$ is the close price at time *t-1*. The true range is then formed as the highest value of the three methods (Equation A.20). Since the first price in a financial time series does not have a previous price, the first true range is simply calculated using method$_1$.

$$Method_1(t) = p_{high}(t) - p_{low}(t) \tag{A.17}$$

$$Method_2(t) = \left|p_{high}(t) - p_{close}(t-1)\right| \tag{A.18}$$

$$Method_3(t) = \left|p_{low}(t) - p_{close}(t-1)\right| \tag{A.19}$$

$$TR(t) = \max_i Method_i(t) \tag{A.20}$$

Next, the average true range (ATR) is calculated. The first *N*-period average true range $ATR_N(0)$ is calculated as the average of the first *N* TRs (in the look-back period *N*) using Equation A.21. Subsequent average true ranges are calculated by multiplying the previous *N*-period average true range $ATR_N(t-1)$ by *N-1*, followed by adding in the current true range $TR(t)$ and finally dividing the total by *N*. This is shown in Equation A.22.

$$ATR_N(t) = \frac{1}{N}\sum_{t=0}^{N-1} TR(t-i) \tag{A.21}$$

$$ATR_N(t) = \frac{[ATR_N(t-1)\times(N-1)]+TR(t)}{N} \tag{A.22}$$

The last step is to calculate the normalized volatility indicator (NVI) by dividing the current average true range $ATR_N(t)$ with the current closing price $p_{close}(t)$ and multiplying the total by 100, as shown in Equation A.23.

$$NVI_N(t) = \frac{ATR_N(t)}{p_{close}(t)} \times 100 \tag{A.23}$$

Two normalized volatility indicators (NVI) were calculated with different period lengths and used as indicators in this paper. The first NVI was calculated using a 10-minute period and the second using a 20-minute period. The two indicators are shown in Figure A.6. The 10-minute NVI is plotted in red and the 20-minute NVI is plotted in blue.

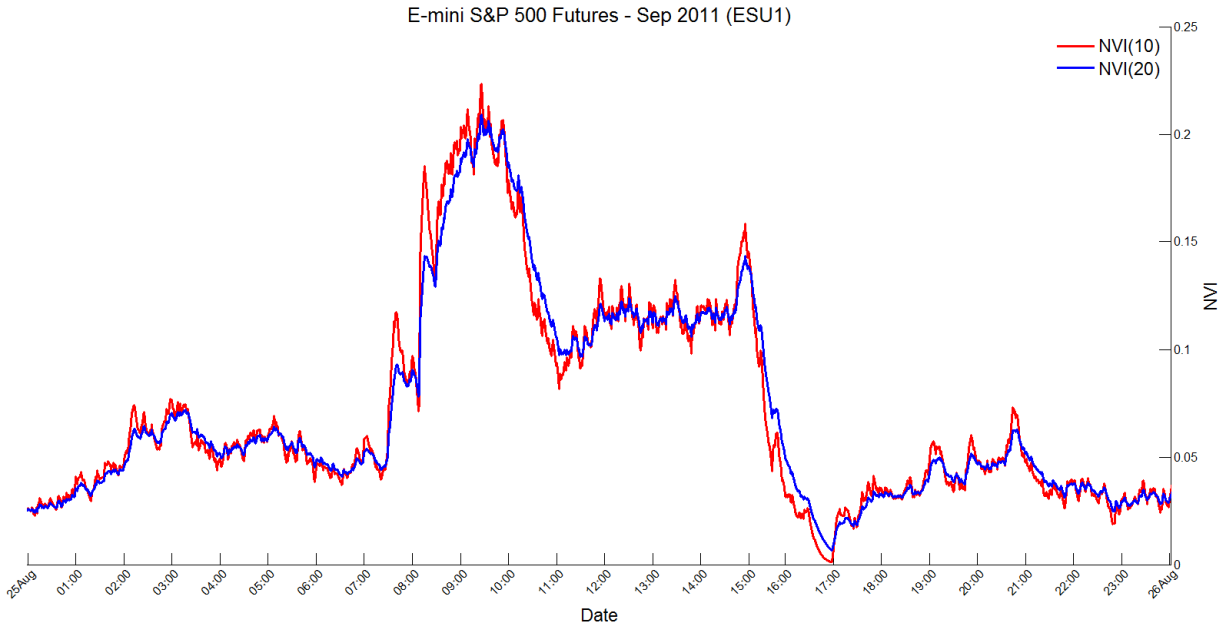

Figure A.6 NVI for the E-mini S&P 500 September 2011 contract (25th of August)

Chaikin Money Flow (CMF)

The Chaikin money flow (CMF) indicator is an oscillator that uses both price and trade volume to measure the money flow volume (MFV) over a specific period *N*. It oscillates round its zero line and is used as an indicator for buying and selling pressure.

The CMF indicator is calculated in two steps. First, the money flow volume is calculated using Equation A.24, where $p_{close}(t)$ is the current closing price, $p_{high}(t)$ is the current high price, $p_{low}(t)$ is the current low price and $volume(t)$ is the current traded volume. The CMF is then calculated as the MFV for the period *N* divided by the volume for the period *N*, as shown in Equation A.25.

$$MFV(t) = \frac{[p_{close}(t) - p_{low}(t)] - [p_{high}(t) - p_{close}(t)]}{p_{high}(t) - p_{low}(t)} \times volume(t) \tag{A.24}$$

$$CMF_N(t) = \frac{\sum_{i=0}^{N-1} MFV(t-i)}{\sum_{i=0}^{N-1} volume(t-i)} \tag{A.25}$$

A CMF indicator using a 20-minute period is displayed in Figure A.7 (upper plot) together with the traded volume (lower plot).

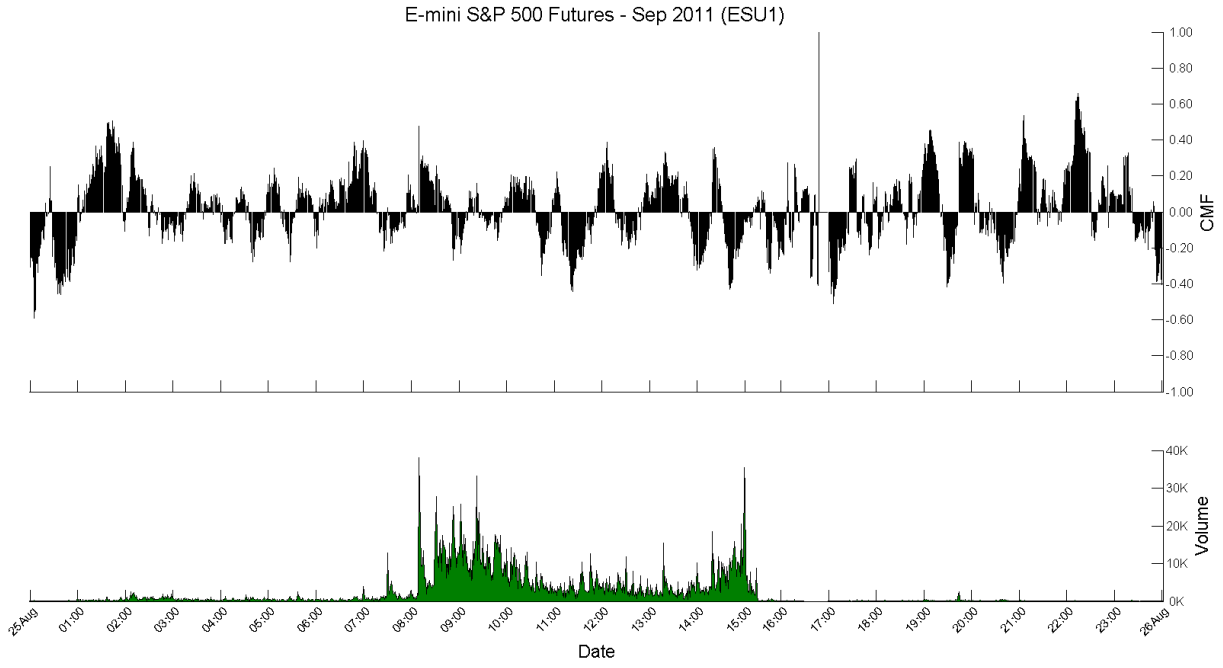

Figure A.7 CMF (top) and volume (bottom) for the E-mini S&P 500 September 2011 contract (25th of August)

Bollinger Bands %B (%B)

The %B indicator is derived from Bollinger bands, which are used as a price chart overlay in order to give a visual queue of the volatility in the market. Bollinger bands are calculated as a combination of a simple moving average (SMA) and volatility (standard deviation). The SMA produces a smoothed price curve and the two volatility bands are offset by a number of standard deviations above and below the SMA respectively. The two volatility bands widen and narrow around the SMA band as the volatility in the market changes.

The standard deviation for the closing price is calculated by using Equation A.28, where $s_N^2(t)$ is the variance (Equation A.27), $\mu_N(t)$ is the mean (Equation A.26), $p_{close}(t)$ is the closing price and N is the period length.

$$\mu_N(t) = \frac{1}{N}\sum_{i=0}^{N-1} p_{close}(t-i) \tag{A.26}$$

$$s_N^2(t) = \frac{1}{N-1}\sum_{i=0}^{N-1}[p_{close}(t-i) - \mu_N]^2 \tag{A.27}$$

$$s_N(t) = \sqrt{s^2(t)} \tag{A.28}$$

The middle band is simply calculated as the N-period simple moving average $SMA_N(t)$ of the closing price (Equation A.29). The upper band is then created by adding the N-period standard deviation $s_N(t)$ of the closing price multiplied by an integer M (Equation A.30). Similarly the lower band is created by subtracting the N-period standard deviation $s_N(t)$ of the closing price multiplied by an integer M (Equation A.31).

$$B_{middle_N}(t) = SMA_N(t) = \frac{1}{N}\sum_{i=0}^{N-1} p_{close}(t-i) \tag{A.29}$$

$$B_{upper_{N.M}}(t) = SMA_N(t) + s_N(t) \times M \tag{A.30}$$

$$B_{lower_{N,M}}(t) = SMA_N(t) - s_N(t) \times M \tag{A.31}$$

The %B indicator measures the current closing price relative to the upper and lower Bollinger bands and is used to identify overbought and oversold levels. It has a value of higher than 1 when the current price is above the upper band, a value lower than 0 below the lower band and a value of 0.5 at the middle band. The indicator is calculated using Equation A.32.

$$\%B_N(t) = \frac{p_{close}(t) - B_{lower_{N,M}}(t)}{B_{upper_{N,M}}(t) - B_{lower_{N,M}}(t)} \tag{A.32}$$

The upper graph in Figure A.8 shows the closing price (blue plot) together with Bollinger bands (red plots) calculated using a 20-minute SMA (green plot) and 2 units of standard deviation. The lower graph shows the corresponding %B indicator. As can be seen in the figure, the upper and lower Bollinger bands widen and narrow around the closing price's simple moving average as the volatility in the market changes.

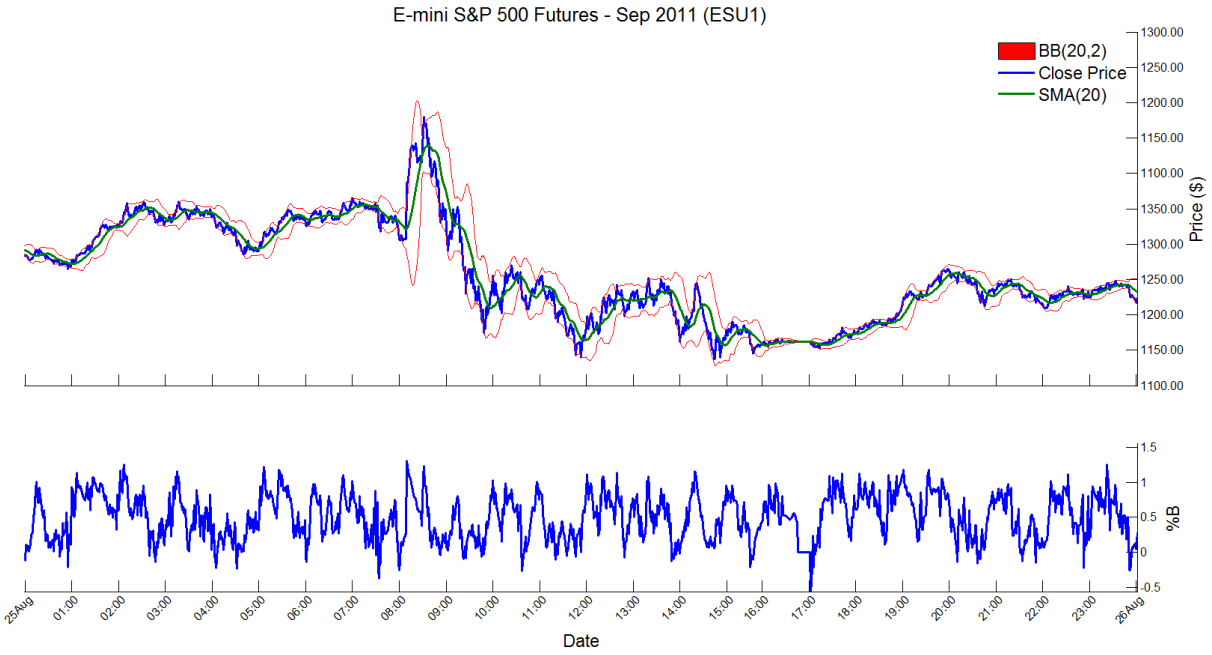

Figure A.8 %B (bottom) and BB bands (top) for the E-mini S&P 500 September 2011 contract (25th of August)

Rate of Change (ROC)

The rate of change (ROC) indicator oscillates around a zero line and shows the velocity in price changes by relating the current price level to the price level *N* time steps ago. Buy and sell signals are produced as the indicator crosses its zero line, where extreme values indicate overbought and oversold levels.

The indicator is calculated as the difference between the current closing price $p_{close}(t)$ and the closing price *N* time steps ago $p_{close}(t\text{-}N)$, followed by dividing by the closing price N time steps ago and multiplying the total by 100, as shown in Equation A.33.

$$ROC_N(t) = \frac{p_{close}(t) - p_{close}(t-N-1)}{p_{close}(t-N-1)} \times 100 \qquad \text{(A.33)}$$

A ROC indicator calculated using a 12-minute period is shown in Figure A.9.

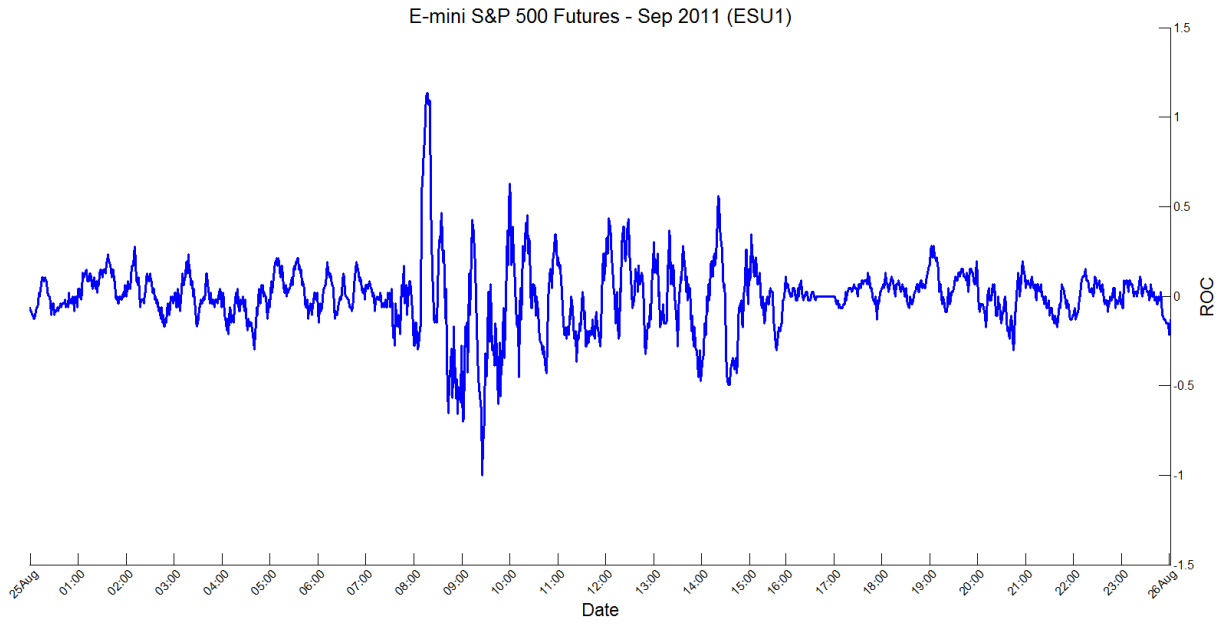

Figure A.9 ROC for the E-mini S&P 500 September 2011 contract (25th of August)

Fast Stochastic Oscillator (%K)

The fast stochastic oscillator (%K) is a momentum oscillator, indicating the current closing price level relative to the lowest low price during the look-back period *N*. Just as the Wn%R indicator, the %K indicator is normalized, but operates in the range [0, 100] instead, and is used for identifying overbought and oversold market conditions. The indicator is calculated using Equation A.34, where $p_{close}(t)$ is the current closing price, $p_{HighestHigh}(0 \leq t < N)$ is the highest high price during the look-back period *N* and $p_{LowestLow}(0 \leq t < N)$ is the lowest low price during the look-back period *N*.

$$\%K_N(t) = \frac{p_{close}(t) - p_{LowestLow}(0 \leq t < N)}{p_{HighestHigh}(0 \leq t < N) - p_{LowestLow}(0 \leq t < N)} \times 100 \qquad \text{(A.34)}$$

Figure A.10 shows a %K indicator (red plot) using a 14-minute look-back period.

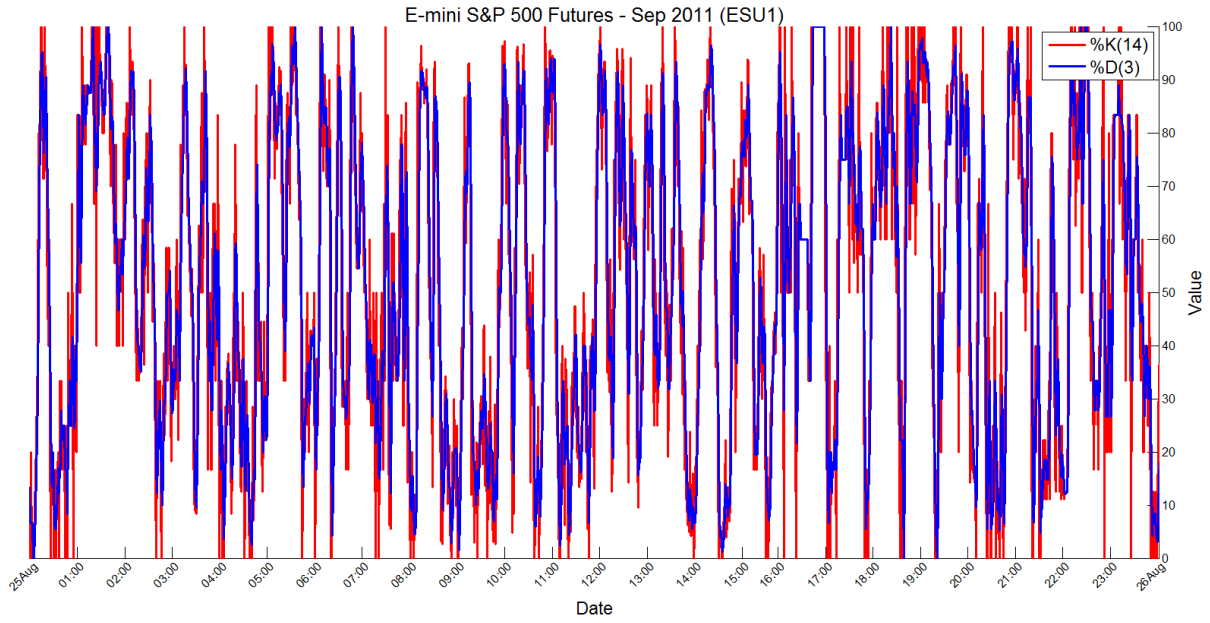

Figure A.10 %K and %D for the E-mini S&P 500 September 2011 contract (25th of August)

Fast Stochastic Oscillator Signal Line – %D(3)

The fast stochastic oscillator signal line (%D) acts as a trigger for the fast stochastic oscillator (%K). It is calculated as an *N*-period simple moving average (SMA) of the %K indicator as shown in Equation A.35.

$$\%D_N(t) = \frac{1}{N}\sum_{i=0}^{N-1} \%K_N(t-i) \qquad \text{(A.35)}$$

A fast stochastic oscillator signal line (%D) using a 3-minute simple moving average (SMA) of the %K indicator is shown in Figure A.10 (blue line).

Högskolan i Borås är en modern högskola mitt i city. Vi bedriver utbildningar inom ekonomi och informatik, biblioteks- och informationsvetenskap, mode och textil, beteendevetenskap och lärarutbildning, teknik samt vårdvetenskap.

På **institutionen för data- och affärsvetenskap (IDA)** har vi tagit fasta på studenternas framtida behov. Därför har vi skapat utbildningar där anställningsbarhet är ett nyckelord. Ämnesintegration, helhet och sammanhang är andra viktiga begrepp. På institutionen råder en närhet, såväl mellan studenter och lärare som mellan företag och utbildning.

Våra **ekonomiutbildningar** ger studenterna möjlighet att lära sig mer om olika företag och förvaltningar och hur styrning och organisering av dessa verksamheter sker. De får även lära sig om samhällsutveckling och om organisationers anpassning till omvärlden. De får möjlighet att förbättra sin förmåga att analysera, utveckla och styra verksamheter, oavsett om de vill ägna sig åt revision, administration eller marknadsföring. Bland våra **IT-utbildningar** finns alltid något för dem som vill designa framtidens IT-baserade kommunikationslösningar, som vill analysera behov av och krav på organisationers information för att designa deras innehållsstrukturer, bedriva integrerad IT- och affärsutveckling, utveckla sin förmåga att analysera och designa verksamheter eller inrikta sig mot programmering och utveckling för god IT-användning i företag och organisationer.

Forskningsverksamheten vid institutionen är såväl professions- som design- och utvecklingsinriktad. Den övergripande forskningsprofilen för institutionen är handels- och tjänsteutveckling i vilken kunskaper och kompetenser inom såväl informatik som företagsekonomi utgör viktiga grundstenar. Forskningen är välrenommerad och fokuserar på inriktningarna affärsdesign och Co-design. Forskningen är också professionsorienterad, vilket bland annat tar sig uttryck i att forskningen i många fall bedrivs på aktionsforskningsbaserade grunder med företag och offentliga organisationer på lokal, nationell och internationell arena. Forskningens design och professionsinriktning manifesteras också i InnovationLab, som är institutionens och Högskolans enhet för forskningsstödjande systemutveckling.

HÖGSKOLAN I BORÅS
VETENSKAP FÖR PROFESSION

BESÖKSADRESS: JÄRNVÄGSGATAN 5 · POSTADRESS: ALLÉGATAN 1, 501 90 BORÅS
TFN: 033-435 40 00 · E-POST: INST.IDA@HB.SE · WEBB: WWW.HB.SE/IDA