Classification of Trading Strategies for the Early Detection of Poor Traders

Daniel Jason Williams

Department of Computing and Mathematics

Faculty of Computing, Engineering and Science

University of South Wales

A submission presented in partial fulfilment

of the requirements of the University of South Wales

for the degree of Doctor of Philosophy

December, 2017

Acknowledgements

I would like to thank my enthusiastic supervisors, Professor Paul Roach, Dr John Wyburn and Professor Hugh Coombs for the guidance, encouragement, insightful questions and advice they have provided throughout my time as PhD student. I have been extremely lucky to have supervisors who cared so much about my work, and who responded to my questions and queries so promptly.

I am very fortunate to have received so much support, advice, data and training from OSTC (Wales), the industrial partner and part sponsor for the project described in this thesis. I would like to thank Michael Shirley especially for his time, ideas, expertise and the trading knowledge that he has given me, which has proved invaluable.

I gratefully acknowledge the support, knowledge and access to high performance computing given to me by HPC Wales.

Finally, I would like to thank the Engineering and Physical Sciences Research Council for providing their part of the funding which allowed me to undertake this research.

Abstract

Successful trading in modern markets depends on the ability to monitor and anticipate changes in exchange rates, share, bond and derivative prices, and to make effective decisions to buy, sell, withdraw or hold based on these changes. Rapid changes in market conditions necessitates consistent but flexible strategies to inform, evaluate and take these decisions. The potential of Artificial Intelligence (AI) based systems to augment, and where necessary, replace human decision-making is clear in this field.

While AI-based methodologies for trading and successful decision-making are discussed in academic publications, the tools (whether "black box" and producing output by "hidden" algorithmic or statistical means, or "white box" and employing explicit causal mechanisms) are typically proprietary, i.e. exact algorithms, heuristics and modelling archetypes employed are not in the public domain. Hence, for the trader, bespoke development is required to apply these methodologies to provide the information advantage sought.

The work described in this thesis derives from an industrial collaboration between the University of South Wales and OSTC Wales Ltd., a trading company specialising in the trading of interest rates, commodity futures and other derivatives. The thesis addresses the exploitation of market behaviour by combining artificial intelligence with financial trading heuristics, to design and produce a working prototype system which quantifies and categorizes trading strategies.

This thesis therefore details the design of, and examines the further implications and applications for, an early warning detection system for identifying 'bad' traders using machine learning. It is assumed traders can be evaluated using the same performance metrics as the trading strategies they employ. That is, a good trader is one that uses a successful strategy. The system is "white box" and hence amenable to analysis and practical experimentation.

The system identifies a set of criteria by which the success of a trading strategy may be judged. It examines these criteria in order to "score" the traders employing these. The trading company, proprietary or otherwise, employing the traders can therefore make better decisions about the amount of funds to allocate to each trader.

The methodology is intended for use by those who manage traders, but can also be used by traders themselves. The detection system found may be used by other concerns in the financial sector whereas the principles examined can be extended to other areas in which decision-making is critical. Examination of the finished system and the process of its composition will provide useful material for academic analysis.

Table of Contents

Chapter 1 – Introduction

Financial markets are an essential part of the global economy as they are a centralised place to exchange assets and other financial instruments. Financial markets attract all sorts of participants such as buyers, sellers, investors, hedgers and speculators. Financial traders are speculators that provide liquidity to the market and they help discover the true value of an asset or financial instrument. Traders may partake in the markets for the thrill or for monetary gain, but to become consistently profitable in this highly competitive endeavour, the trader must commit to continuous learning in an ever evolving environment. Profits can be extracted from the financial markets in many ways but there is a catalogue of things a trader must become knowledgeable about including risk, psychology, capital management, how to create and apply a trading strategy and to how to recognise if their trading strategy has an edge in the markets.

The primary aim of this thesis is to design an early warning detection system for bad traders using machine learning. Traders can be evaluated using the same metrics as trading strategies. Thus, this research will focus on classifying trading strategies rather than focusing on real life traders (which may infringe personal and proprietary information). Finding profitable trading strategies in ever-evolving markets is difficult but is crucial for the continued success of a trader. To test our early warning system for bad traders, the trading strategies used to test our system need to contain 'good' as well as 'bad' trading strategies.

From the operations that take place at OSTC, there are two main methodologies that are being employed by traders; these are technical analysis (1.3.3) and fundamental analysis (1.3.4). Both categories of analysis can be employed to provide information advantages in trading. However, as will be explained later, the former is quantitative, whereas the latter is both qualitative and less immediate. This thesis addresses the broad principles of the possibility and means of categorizing traders. To meet the primary aim of the work, models of traders must be built, and this is facilitated by quantitative technical analysis algorithms and real-world financial data. Therefore the scope of work has been restricted to technical analysis. This effectively describes the second aim of this thesis, which is the creation of

trading strategies, the models of traders, through combining the buy and sell signals from technical analysis algorithms.

It should be noted that this pragmatic approach to modelling trader and market behaviour contrasts with the typical academic method of assuming ideal (i.e. perfectly-competitive, non-arbitrageous) markets and seeking closed-form solutions relevant only to these. The approach taken here is therefore of direct and practical relevance to the University's industrial partner on this project.

The thesis is divided into five sections, beginning in Part 1 with the background and literature review of machine learning applied to the classification and creation of trading strategies, Chapter 2. In Part 2, the technical analysis interpretations that are implemented in this thesis are outlined in Chapter 3. An overview of the performance metrics that can be obtained from technical analysis interpretations and trading strategies is then provided, Chapter 4. The extent to which historical performance metrics can influence future performance metrics is also investigated in Chapter 5. In Part 3, an attempt is made to create trading strategies using a Genetic Algorithm in Chapter 6 and 7, optimising combinations of technical analysis algorithms to generate a pool of models of traders. However, the means of identifying an appropriate fitness function was unclear, and the approach may have suffered from overfitting. Having established the difficulties with determining useful fitness functions, this work informed the subsequent choice of a machine learning method that determines the fitness function as its output – the Adaboost algorithm. In Part 4, experiments are conducted using the Adaboost algorithm as the underlying classification algorithm to create classification systems that attempt can classify and identify 'bad' traders. In Chapter 8, classification systems trained using the performance metrics of single technical analysis interpretations are compared. In Chapter 9, imbalances in the training validation and outsample dataset are discussed. Classification systems trained using the performance metrics of trading strategies using a number of technical analysis interpretations are compared. In Chapter 10, classification systems are trained using the performance metrics of trading strategies using multiple market datasets. In Chapter 11, the categorization of what makes a trading strategy 'good' or 'bad' is changed and the classification systems are trained to predict various performance metrics in the future. In Part 5, 11.7 demonstrates how the Adaboost-based classification systems that are

explored in this thesis can be altered to produce regression output and also shows how to visualise and make sense of the decisions of these systems.

A prototype system has been developed in this project, such that users of the system (managers of trading companies) can create the classification systems developed in this thesis and use the classification of the classification systems to classify traders. The users of the software can also choose their categorisation metric and value; this is described and illustrated in Appendix D.

1.1 Financial Markets and their Prediction

The definitions below for terms commonly used in describing financial markets are sourced from (McLaney, 2006).

An *ordinary share* (or more often, just *share*) is an equity (a part ownership) in the company that issues it. The *primary market* for such shares refers to their purchase when new.

The *secondary market* is the financial market in which previously issued shares and *bonds*, *derivatives*, *options* and *futures* (see below), are bought and sold. Despite its name, in providing ongoing prices for investors, the secondary market is the primary means of evaluation of asset prices, and hence of the companies and financial institutions whose assets they represent. It also administrates considerably greater value and volume of trade than the primary market.

Bonds are loans made to companies or governments. They typically pay interest in the form of *coupons* (set payments) and may be redeemable for a face value at some point in the future. The bond market is globally more important than the share markets.

Foreign exchange markets (FX or Forex) are markets that allow for one currency to be converted into another currency. The foreign exchange markets averaged more than $5 trillion per day in April 2016 (Triennial Central Bank Survey of foreign exchange and OTC derivatives markets in 2016, 2016).

Derivatives are contracts to buy or sell shares, bonds, or some other asset, the *underlying*. The most important derivatives are the *future* and the *option*.

A *future* is a contract to trade in the share, on an agreed future date, the delivery date, for an agreed price, the futures price. The holder may either retain the future, committing to buy or sell the underlying, or sell the contractual obligation on to a third party. The holder's decision to do so depends on the behavior of both the underlying share price and the current future price.

Of particular interest to the present study is the *interest rate future*, in which an interest-bearing instrument such as a bond is the underlying asset. The *interest rate derivatives market* is currently the largest such market.

An *option* on a share is a right to trade in the share, on an agreed future date, the exercise date, for an agreed price, the exercise price. The holder of the option pays the option price for this right, and may either retain the option pending the decision to exercise, or sell the option on to a third party. The holder's decision to do so depends on the behavior of both the underlying share price and the current option price. For instance, the call option gives the holder the right to buy a certain number of shares, and the holder is likely to exercise if the share price falls below that of the exercise price. Alternatively, if the option price itself rises sufficiently, the right to buy may itself be sold on; this is likely to follow a change in share price. A put option gives the holder the right to sell a certain number of shares.

The share price's *volatility*, the tendency of the price to change over different time scales, is taken to be a measure of the risk attached to trading in the share and in its options. Against the risk, the possible profit or loss, the return, must be estimated.

The *risk* of an investment is the probability that the return will be different from the expected return.

A *trading strategy* is a policy of trade typically having the aim of optimizing return while minimizing risk. Such a strategy amounts to a decision-making protocol. The most important decisions are whether to buy, to sell, or to hold, when to buy, and when to sell. The correct choice of trading strategy with respect to an option therefore depends sensitively on anticipating the behavior of share prices. Assuming a long position, in which assets are held for sale, it is necessary to have sufficient confidence that prices are now lower than they will be, in order to buy. Given a short position, in which assets are bought for sale at an agreed

price, it is necessary to have sufficient confidence that prices are now higher than they will be, in order to sell.

To buy a spread is to buy an option at a given premium (price) and write a second option with the same exercise date, a lower premium and a different exercise price. The numbers are chosen to create a position in which a profit will be made whether or not the options are exercised. If the investor exercises the option, it is because profit can be made on the resale of the underlying asset. If the written option with the lower premium is exercised, the investor will profit on sale of the underlying.

To sell a spread is to write an option for a higher premium than one with the same underlying that one buys, e.g. to write a put option after buying a similar put option (perhaps with a closer exercise date). If either option is exercised, a profit is made by selling the option at a higher premium than was paid for the other.

A *proprietary trading company* trades assets with its own money so as to make a profit for itself. An example of such a company is the industrial partner for the project described in this thesis, OSTC Wales, Ltd., based at Swansea. This company's traders must make fast, critical buying and selling decisions in response to rapidly changing markets involving many competing traders. Market indicators, determined through the monitoring of past and current prices, can be used to construct trading strategies and to determine optimum ranges for setting buy and sell prices; these rules form the basis of *trader behavior*. The requirement for fast decisions runs the risk of traders making poor trades, which is a threat to the profitability of the traders and their employers.

All market datasets used in this thesis are Foreign Exchange (FX) rate markets. The focus on these markets is justified by the volume of trading within them. It is assumed here that analyses performed on these markets and methodologies developed for them will be indicative of approaches likely to be successful in other financial markets. The early warning system described in this thesis functions as a decision support tool, intended to give OSTC an information advantage over other competing proprietary trading companies and other market participants. It can also be employed to assist trader discipline.

1.2 The Industrial Partner: OSTC Ltd

OSTC have offices in many countries, including the United Kingdom, Poland, Portugal, Spain, India and Mauritius. The company specialises in the trading of interest rates, and of commodity futures and other derivatives, and each office trades in a slightly different way.

The majority of markets are traded electronically via exchanges across the internet. OSTC Swansea trade the markets via these exchanges using the software 'Stellar' and 'Stellar spread machine'. The 'Stellar spread machine' gives traders the ability to trade different contract dates for a particular asset. Interestingly, as they are based on the same asset these spreads are interlinked and price discrepancies can occur.

Spread matrices which are used in the 'Stellar spread machine' software have some interesting properties. A trader can buy two consecutive spreads, for example, the March to June spread and the June to September spread, each of which spans a three month contract for a particular asset. The trader has effectively bought a six month spread at the price of each of the individual three month spreads. If the six month contract was priced higher than the price paid for each of these three month contracts then the trader has paid a lower price for the six month contract. As another example, consider that a trader can buy a six month spread, for instance, the March to September spread, then sell either of the three month spreads.

Approximately 75% of the markets OSTC trade in are commodities (gas, oil, Brent crude, natural gas and light sweet crude) and other financial derivatives (FX and equity futures), and the other 25% of markets are short term interest rates (OSTC, 2012).

OSTC traders attempt to forecast markets by using *technical analysis* (1.3.3) which aims to model and exploit human population behaviours, and *fundamental analysis* (1.3.4), which examines company and broader data directly. The latter data includes economic data such as the gross domestic product (GDP) which measures how quickly an economy is growing and the consumer price index (CPI) which is a popular measure of inflation in retail goods services.

Markets are driven by traders across the world and without buyers and sellers, markets would not exist. To the extent to which the behaviour of groups of people is predictable, the

market is predictable. Traders consider the actions that other buyers and sellers might make when the market changes in order to take advantage of their situation and gain from another buyer's or seller's decision.

1.3 Finance and Trading

Financial exchanges are sources as well as measures of increasing wealth and there have been many attempts to predict the future price value of an asset. Determining an asset's fair price, and forecasting its future price direction, is problematic in that the price is a complex, ever evolving system with many different dependent variables to consider. These variables include several from areas not discussed in thesis, such as high frequency trading and insider information. Although there are many different methodologies used to predict the price direction, this thesis will only use technical analysis models and only briefly discuss other areas.

Unless otherwise noted, all figures in this chapter have been produced by a bespoke application developed for this project, or in Microsoft Excel. The function and design of the former is explained in the subsequent chapters.

1.3.1 Traders

Each individual market participant (trader, investor, hedge fund etc.) has one main objective: to make money by following the "buy low and sell high" rule. The difference between the prices (minus some transaction cost) is the profit of a trade. The rule also means "sell high then buy back low", and "buy in anticipation of a higher price and sell in anticipation of a lower price" (Lim, 2015). Generally traders employ, but are not limited to, two methodologies for determining whether an asset is worth buying or selling. These are *technical analysis* (1.3.3) which analyses past and present price, volume, trends, momentum and similar market information, and *fundamental analysis* (1.3.4) which analyses the company and the market to which the company belongs. Both areas can and are used together and along with other methodologies to make more informed decisions. Traders and other market participants are always seeking innovative ways to acquire an 'edge' (an advantage over other participants) and thereby make money from the market.

Traders deploy trading strategies which consist of many components with the aim of making a profit. Some traders automate their trading strategies by implementing them in code. This

removes human emotion, bias and error and allows the automated trading strategy to be objectively tested on past, similar and artificial market data. Automated trading strategies however may contain errors in the code or not fully behave as intended. Prices from the brokerage platform may report a drastically incorrect bid and offer price, the automated trading strategy may perceive the asset to be undervalued and allocate 100% of the capital or even worse become overleveraged.

The flash crash on May 6, 2010 saw stock indices drop significantly and recover in the space of 36 minutes (Treanor, 2015). Though high frequency trading strategies did not cause the problem, high frequency trading strategies exacerbated the problem by further selling in response the falling market (Kirilenko, Kyle, Samadi, & Tuzun, 2017).

1.3.2 Market Data

Instead of working with raw numbers, market data are presented to the trader graphically (*e.g.* as in Figure 1).

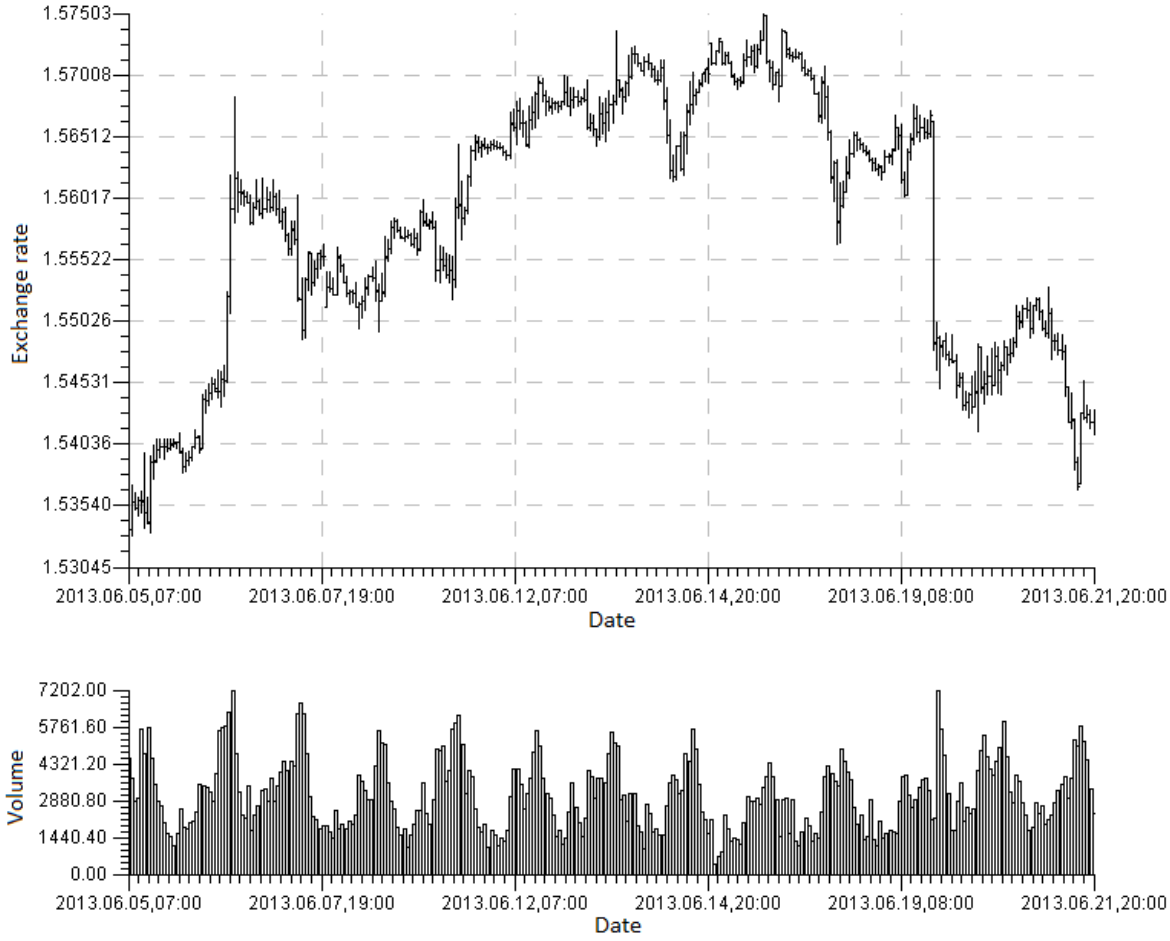


Figure 1: Market data, hourly GBPUSD exchange rates and volume of transactions against time

Prices are represented by 'candlesticks' which show the open, high, low and close price over a specific period of time, as illustrated in Figure 2. Market data can be viewed at various timeframes so each candlestick could represent trading activity over a minute, hour, day, week *etc.* For daily market data, each candlestick outlines what the price action was over a particular day. The open and close prices denote the price at the start and the end of the day respectively and the high and low prices denote the highest and lowest price of the day respectively. Figure 2 shows two variations on the candlestick representation; the second representation is coloured in red or green depending on the open and close prices. If the open is above the close then the colour of the candlestick is red and if the open is below the close then the colour of the candlestick is green.

Volume is normally represented as a bar graph underneath price data. Volume denotes the number of transactions which took place for a given candlestick. This helps the trader gauge

how liquid is the market (high liquidity corresponds to a large number of buy and sell offers) and gauge the amount of activity for each candlestick (for a daily chart, the amount of daily activity).

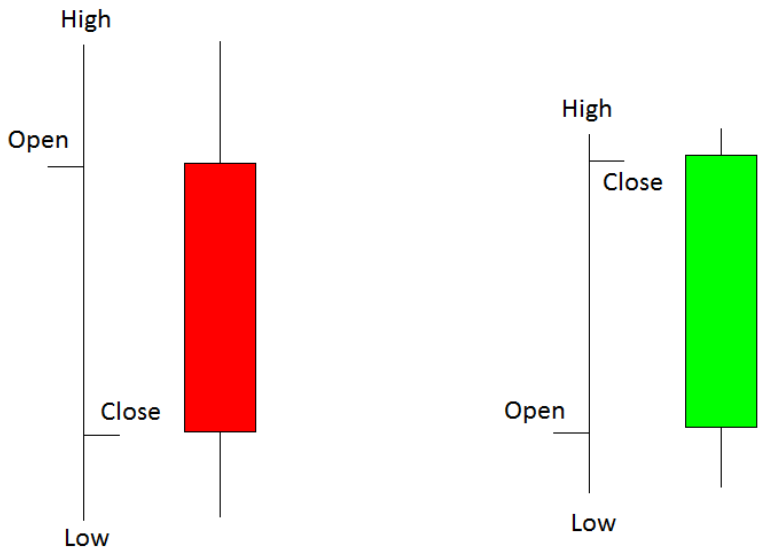


Figure 2: Visual representations of a candlestick with and without colour. Each candlestick indicates open, high, low and close prices.

1.3.3 Technical Analysis

Technical analysis uses market data, primarily price and volume data, to predict an asset's future price movement, where 'asset' can refer to *e.g.* a stock, bond, commodity *etc.* (Kirkpatrick & Dahlquist, 2010). Technical analysis models attempt to model some observed behaviour or dynamic shown by the market. Technical analysis is normally shown alongside market data or is an overlay upon market data. The benefit of portraying this information visually is that it helps the trader to quickly and clearly identify key information about the model, such as turning points or trends. Additionally it is possible to see this information relative to other information such as price, volume and other technical analysis models.

All information about an asset and future expectations is assumed to be reflected in the price (Aronson, 2011). Technical analysis assumes that the price action and dynamics of the market tend to repeat themselves. Technical analysis is seen as a self-fulfilling prophecy

(Jordan, 2014); as participants make buy and sell decisions based on these models, the price becomes influenced by the models.

One of the main advantages of technical analysis is that it can be applied to any market and timeframe. Unfortunately the interpretation of technical analysis models is highly subjective and there always exists an opposite interpretation. For example, consider a technical analysis indictor such as the stochastic oscillator, which is a momentum model that indicates market direction (described in Section 3.3.3). In identifying direction, it could be interpreted as indicating that an asset is overbought and that the market should revert back to a lower price. On the other hand that same momentum model might indicate that the asset is undervalued and so it is moving to a new 'true' price that is higher. Market participants have conflicting biases, interpretations and behaviours.

In technical analysis, *chart patterns* and *candlestick patterns* attempt to identify the future price direction visually. The *head and shoulders* chart pattern is well known and represents a reversal in the price trend if the price crosses a ‘neck line’ which is indicated by the horizontal red line in Figure 3 (Zapranis & Tsinaslanidis, 2010). The candlesticks visually form a shoulder, a head and a shoulder hence the name head and shoulders.

Figure 3: Hourly EURUSD market data containing the Head and Shoulders pattern with the red neck line

Another area of technical analysis is that of *technical indicators*. Technical indicators are graphical representations of formula. The simple moving average is a technical indicator that attempts to model the trend by taking the average close price of the last n days. One interpretation of the simple moving average is that if the average price over the last n is days is below the current close price then the market is seen to be trending upwards. Another interpretation of the simple moving average is if the moving average's gradient is positive then the price is in an uptrend and if the gradient is negative then the price is in a downtrend.

The *stochastic oscillator* is another technical indicator that attempts to model *momentum* (the tendency of a trend to continue). The stochastic oscillator compares each closing price to its price range over the last *n* days. The asset is said to be overbought if the current stochastic oscillator value is above 80% of the range and oversold if below 20% (Figure 4).

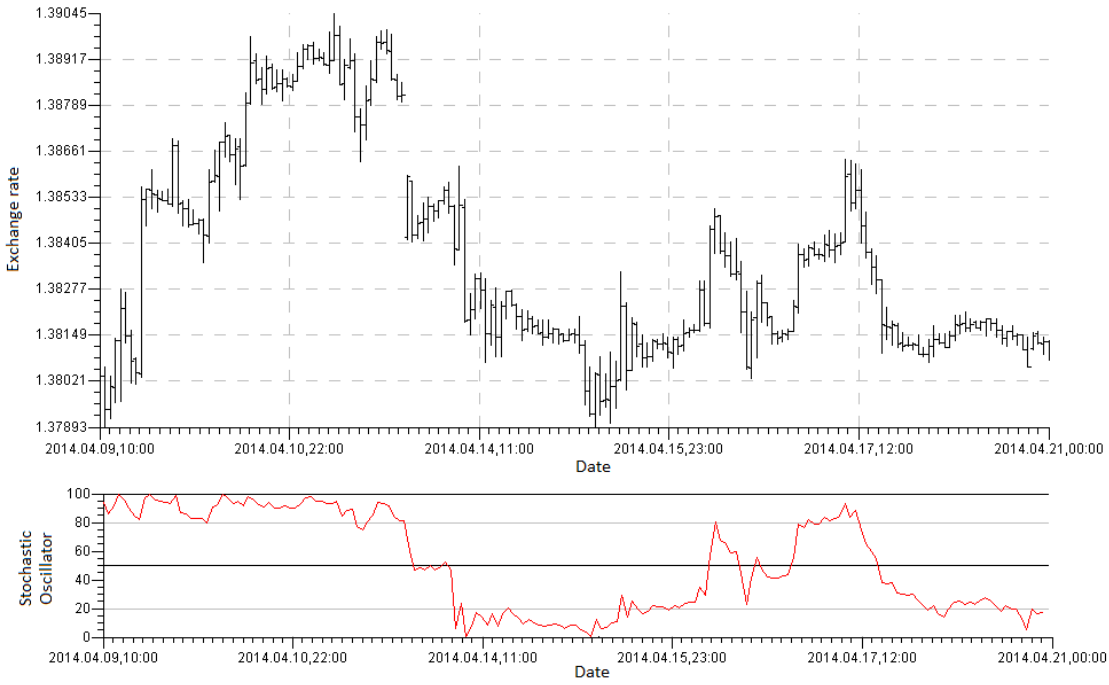


Figure 4: Hourly EURUSD market data with the stochastic oscillator underneath using $n = 80$

Bollinger bands were created by John Bollinger and consist of a moving average, a lower band and an upper band. Both the moving average and standard deviation is calculated by

using the past n prices. The lower band is *k* times the standard deviation below the moving average and the upper band is *k* times the standard deviation above the moving average. If the asset price travels above the upper band then the asset is said to be overbought and the price will fall (Figure 5) (Lui & Zheng, 2011). However the opposite is also true, if the price is above the upper band then it is said to be trending upward. Standard deviation is a proxy for volatility so narrow bands indicate that there is less price volatility and wide bands indicate more price volatility.

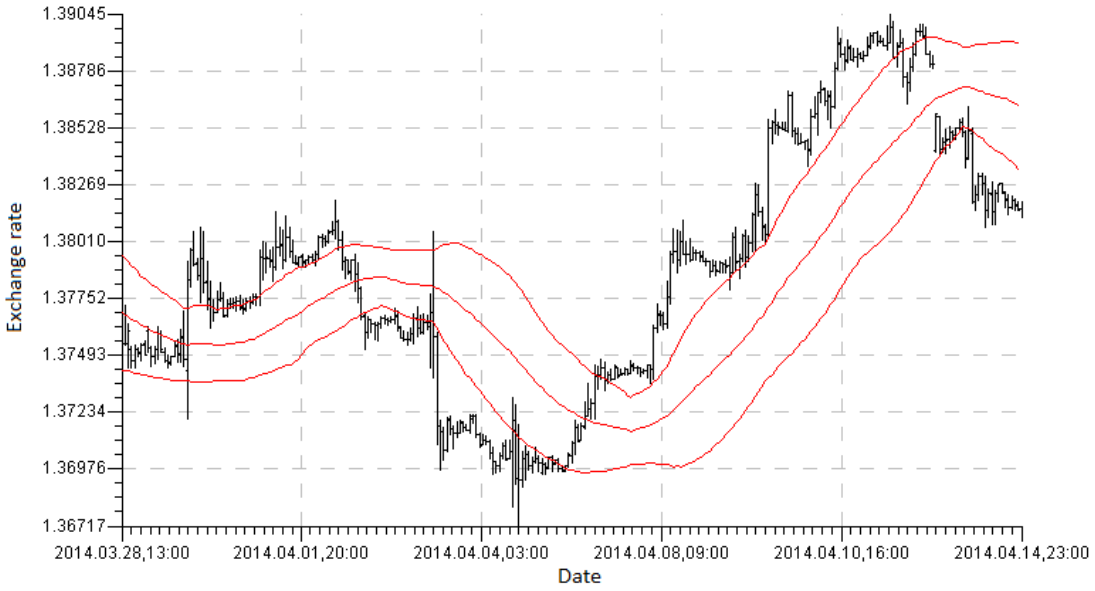


Figure 5: Hourly EURUSD market data with an overlay of a Bollinger band using $n = 60$

In technical analysis there are many ways of interpreting the different models, and some may be more predicative than others. Combining technical analysis models is common as more informed decisions can be made however using too many models and adjustable parameters variables can create a trading system that produces too few trades and depending on the trading strategy creation process, fit historical data and have no predictive power on future prices.

1.3.4 Fundamental Analysis

Fundamental analysis is mainly concerned with finding the *intrinsic* or *true value* of an asset by analysing industry trends, related economic and financial data (Khanifar, Hossein, Jamshidi, & Mohammadinejad, 2012). This includes daily news events and business performance.

The intrinsic or true value of an asset is the sum of all information including that of future expectation. The intrinsic value is a somewhat theoretical concept, which leads to an assumption that profits from price fluctuations above and below the true value of an asset can be obtained if one subscribes to the strategy that the price will always move towards its true value (Bartram & Grinblatt, 2017). If an asset is mispriced lower than the asset's intrinsic value then the asset is undervalued, whereas if the asset is mispriced higher on the market then the asset is overvalued (Kothari, 2001). Market participants speculate, have biases and do not know all the information about an asset. A proportion of market participants execute trading strategies that are based on different assumptions to fundamental analysis and technical analysis. Finding undervalued and overvalued assets is made harder by not knowing what the market participants will do regardless of what the notational true price of an asset is or will be.

Calculating the intrinsic value is difficult because not all information is available and the evaluation is dependent on the interpretation of the analysist evaluating the asset. For instance, suppose an executive, thought to be the main innovator and visionary of a company, left for a rival company. The company's share price may fall because it would be perceived that the company had lost future potential for profit. Whether or not this is fair depends on the correctness of the perceived importance of that person, a subjective matter of opinion.

There are two approaches to evaluating an asset; the *top-down* and the *bottom-up* (Sharpe W. F., 1999). The top-down approach considers high level and general information like exchange rates, national productivity, inflation *etc*., and then narrows the search to specific sectors or industry, then to specific assets. The bottom-up approach starts with a specific asset, regardless of the industry or sector. The analyst looks at low level information; for instance, if the analyst was evaluating a company then he/she would look at the company's financial reports and accounts.

Fundamental analysis relies on accurate reporting and credible accounting. Tesco, a FTSE100 company, overestimated their profits in its half-year profit guidance by £250 million (BBC, 2014). Market participants lost trust in Tesco and this news lead to an 11.6% fall in the share price.

Academically, the fair price is usually equated with the expected value under fair market conditions. This interpretation need not be addressed outside any academic publications resulting from thesis study.

1.3.5 Mathematical Analysis

Mathematically, the volatility (risk) of an asset has been typically modelled as a function of statistical variance over a given time. However no model has evaluated risk perfectly, making too many, or unrealistic, assumptions. Volatility has always been a problem, due to factors typically divided into *intrinsic* (or *specific*) and *extrinsic* (or *market*) risks. Extrinsic risks include natural disasters, macroeconomic factors and political news that could instantly change the market behaviour. These events are hard to predict and it can be hard to determine what they mean in the case of a particular market. For instance, global financial perceptions changed dramatically when the Lehman Brothers' bank faced bankruptcy (Wearden, Teather, & Treanor, 2008).

Companies whose business models depend on risk assessment are especially vulnerable. The Long Term Capital Management Company was a hedge fund management firm which thought it had a winning model of the markets. The company's board of directors included the Nobel Prize winners Myron Scholes and Robert Merton, noted for their contribution to Economic sciences. The company made profits outperforming other similar companies until the East Asian financial crisis and then the Russian financial crisis which resulted in the company's collapse. The models the company used did not account for such supposedly improbable events.

Mathematicians have also tried to formulate the true price value of assets with a more abstract approach, using equations like that of Black Scholes and the related binomial options pricing model of Cox, Ross and Rubenstein.

The Black Scholes equation was formulated by Fischer Black and Myron Scholes (Black & Scholes, 1973) ; it is a partial differential equation which describes the price of the option over time and is based on Brownian motion. The equation attempts to return the true option price and one could hedge with this equation to eliminate risk. The equation has different variations to accommodate different option types and the equation has been extended to accommodate for dividends.

The binomial option pricing model (Cox, Ross, & Rubinstein, 1979) is a discrete-time model algorithm that attempts to return the true option price of a particular option using the probability of upward and downward movements. It was designed to reproduce the Black Scholes formula by discrete means.

All such models are dependent on explicit assumptions which allow useful solutions to be found. However these same assumptions can render the models unrealistic.

1.4 Aims and Objectives

Traders must make fast, critical buying and selling decisions in response to rapidly changing markets involving many competing market participants. Technical analysis interpretations, determined through the monitoring of past and current prices, can be used to construct trading strategies. The primary aim of this project is to construct a decision support system for the classification of traders and thereby the early detection of bad traders. For this, a secondary aim is to generate trading strategies, as models of traders, by combining the buy and sell signals from technical analysis interpretations.

The work of this thesis combines computer science (machine learning and modelling techniques) with quantitative financial trading logic. The involvement of the University of South Wales in the development of the system has provided technical expertise in these disciplines that are not currently available within OSTC Wales.

The objectives are to:

- Identify a set of criteria by which the success or otherwise of a trading strategy may be judged, i.e. the trader who fails to meet these criteria, or a subset of these criteria, can be considered a bad trader.
- Secondarily to score trading strategies so that they can be ranked and compared to other trading strategies. The propriety trading company employing the traders can therefore make better decisions about the amount of funds to allocate to each trader relative to other traders.

While broad methodologies for trading strategy generation and for decision-making in general within Artificial Intelligence, are discussed within academic publications, the tools

('black box' or causal) for supporting financial decision-making are proprietary, and the exact algorithms used for market prediction are not in the public domain. Hence, for the industrial partner, bespoke development is required to compare and combine methodologies and use the outcomes of this thesis to create software which can be exploited by managers at OSTC.

The methodology proposed for classifying traders and trading strategies is the main novel contribution of this work, in that no attempt to achieve this is extant in the literature. For USW, the specific application and the access to trading knowledge within the company has enabled targeted and novel development with the additional element of a grounding and immediate application in the trading community. This practical focus has typically been lacking in academic work to date.

This work has also advanced trading studies generally by:

- Identifying a successful methodology for the classification of traders and trading strategies, incorporating the Adaboost machine learning algorithm;
- Applying this methodology to representative artificial data, demonstrating the effectiveness of the approach;
- Determining a fitness function, weighting standard criteria of trader performance, for evaluating trading strategies that can subsequently be employed in other optimisation tools;
- Constructing a mechanism for identifying bad traders;
- Determining that it is harder to distinguish between good traders and potentially good traders.

Chapter 2 – Machine Learning and its Application to the Classification and Creation of Trading Strategies

This chapter introduces classification methodologies and reviews the use of these methodologies in literature as applied to the classification of traders. It is first noted that while there is much in the literature about the application of machine learning to market forecasting, there appears to be nothing specifically on its application to classifying the performance of traders or trading strategies. This may be because such systems, if they exist, would likely be proprietary.

This chapter will first introduce the commonly used types of learning methodologies in machine learning in 2.1, then discuss classification and regression models in 2.2, then discuss the problems of overfitting and underfitting in 2.3. Several different machine learning algorithms are outlined and their applications to financial trading are discussed in 2.4. Chapters 8, 9, 10 and 11 explore various binary classification systems so discussion of how to evaluate binary classifiers in outlined in 2.5. Lastly experiments were conducted in Chapter 6 and 7 that used Genetic Algorithms to create trading strategies so optimisation and in particular Genetic Algorithms are discussed in 2.7.

2.1 Types of Learning

Supervised learning (Mohri, Rostamizadeh, & Talwalkar, 2012) uses a labelled training dataset, which consists of input features and an output feature(s), to learn a mapping from the input features to the output feature(s). The idea behind supervised learning is that the algorithm can learn from a dataset of correct examples. Supervised learning requires a training dataset to be generated and to include predictive input features that can help predict an output feature(s). Generating a training dataset can be time consuming and could be of limited value. In natural language processing for example, producing a training dataset of part of speech tags (nouns, verbs, pronouns, etc.) is incredibly time consuming to create and only a finite amount of training data can be produced which normally depends on the project's budget.

Unsupervised learning (Ghahramani, 2004) on the other hand does not use a labelled training dataset but attempts to find interesting patterns using only input data. *Clustering* techniques are unsupervised learning techniques that attempt to cluster data into different

label categories. Though unsupervised learning techniques are not limited to a labelled training dataset, they cannot easily evaluate performance in the same way a supervised learning algorithm can using a training dataset. Instead the unsupervised learning algorithm uses some objective or utility function as a proxy for progress in extracting patterns from the data.

Semi-supervised learning (Mohri, Rostamizadeh, & Talwalkar, 2012) combines both supervised and unsupervised learning methodologies to leverage labelled training data with extra unlabelled data to learn a mapping from the input features to the output feature(s).

This thesis will focus on using supervised learning techniques as it is possible to label and create large training datasets.

2.2 Classification and Regression

Artificial Neural Networks and Support Vector Machines (discussed in 2.4.1 and 2.4.2) create regression models that attempt to estimate the relationship between a set of known features and a feature that the machine learner practitioner wishes to predict. These regression models output values that are continuous. Classification models that are produced by algorithms such as Adaboost (discussed in 2.4.3.2) on the other hand, also use a set of features but attempt to categorise the input features into two (or more) categories.

In the intended application of classifying traders, classifiers attempt to categorise the trader into the good or bad category based upon features that are fed into the classifier (such as the trader's profit factor over the past month). Regressors are similar but instead of categorizing traders into class labels, the trader is given a score of how likely the trader is to be either good or bad.

2.3 Overfitting and Underfitting

Overfitting occurs when a machine learning algorithm's learning algorithm begins to memorize the training dataset. An overfitted classifier or regressor is said to be *high variance* and rerunning the machine learning algorithm on a different training dataset would produce a different classifier or regressor. *Low variance* classifiers and regressors are models that do not change much if a different training dataset is used. An overfitted model will report very successful performance on the training dataset but will be suboptimal in

classifying or regressing data outside of the training dataset. To help overcome overfitting, a *validation dataset* can be used to see how the classifier or regressor performs at each iteration of the learning algorithm on a different dataset, see Figure 6.

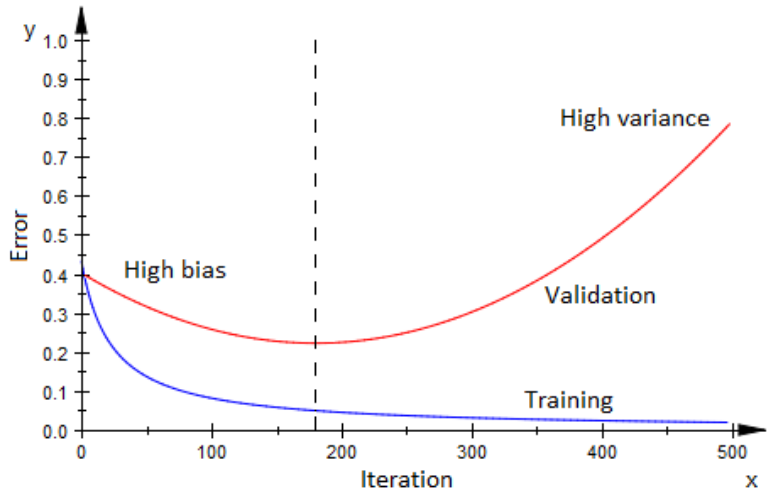


Figure 6: Illustration of how validation data helps to avoid overfitting and underfitting. This also identifies where high bias and high variance exist during the model's construction

Underfitting is where the learning algorithm has not iterated enough to choose a classifier or regressor that has sufficiently captured the general classification in the training dataset. An underfitted classifier or regressor is said to be *high bias* meaning that it suggests more assumptions about the target classification or regression model. *Low bias* suggests fewer assumptions and allows for the target classification or regression model to be complex enough to be able to construct the target model.

There is a clear trade-off between two sources of error, bias and variance, see Figure 6. A model with high bias is unable to capture the general classification or regression of the data as it is too simple and underfits due to the lack of complexity. On the other hand, models with high variance can overfit and memorize the training dataset. Overfitted models lose the ability to generalise and can hence can no longer successfully predict unseen data.

At each iteration of the learning process the model's error can be computed using an unseen dataset set called the validation dataset. For successful learning, the model's error will generally decrease on both the training and validation datasets with increasing iteration. The error derived from the training dataset normally converges as the classifier is

forced into memorizing the training dataset. The error derived from the validation dataset, however, should, at some iteration, change direction and increase. This occurs when the classifier starts to memorize the training dataset. An optimal classifier can be chosen at the iteration before the one at which the error begins to increase on the validation dataset. Practitioners may also observe a sharp increase in error from the classifier on the validation dataset. Such sharp increases suggest that the classifier has memorized the training dataset.

2.4 Machine Learning Algorithms

This section outlines three machine learning classification and regression methodologies, Artificial Neural Networks 2.4.1, Support Vector Machines 2.4.2 and Ensemble methods 2.4.3.

2.4.1 Artificial Neural Networks

Components of *Artificial Neural Networks* will be explained before detailing financial applications that employ them. *Artificial neural networks* (Figure 7, showing a feed forward neural network, as described in 2.4.1.2) were inspired by the neuronal model of the human brain and provide a general framework for solving problems, including classification and regression problems. Artificial neural networks mimic the accepted way in which systems of neurons predict some output value(s) given some input values. These networks can make sense of high-dimensional input data and extract features from this data (Cheh, Weinberg, & Yook, 2011). There are different variations of artificial neural networks and the main concepts are covered in the following sections.

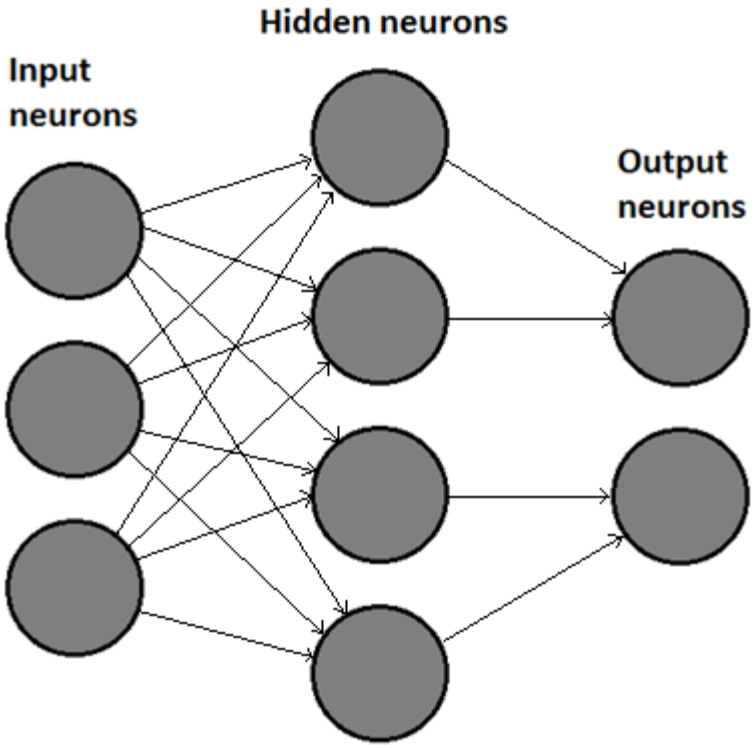


Figure 7: A simple feed forward artificial neural network

2.4.1.1 Single Layer Perceptron

The basic information processing unit in an artificial neural network is the perceptron (Figure 8). Perceptrons are linear classifiers that attempt to linearly separate input data to distinguish one class from another and can also be used to perform regression.

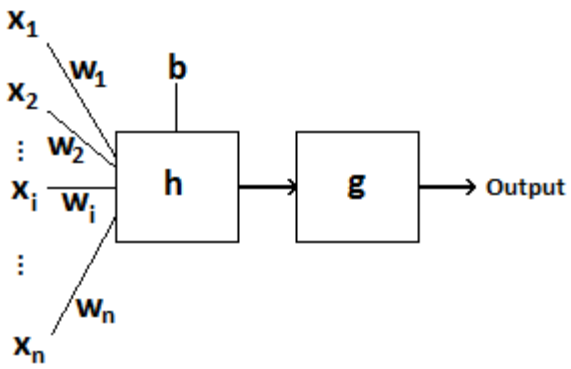


Figure 8: Outline of a perceptron

Here x_i denotes an input value from an input feature, w_i denotes the weight of the x_i input value, b denotes a bias value, h denotes the function $b + \sum_{\forall i} x_i w_i$ and g denotes the activation function.

The weights w_i and the bias value b are adjustable parameters used to create a linear classifier. The values to these parameters are learned by using a perceptron learning algorithm. Each input value x_i is multiplied by an adjustable weight value w_i. The function h sums all of the weighted input values and adds a bias value b. The bias value b is used to adjust the input value into the activation function g:

$$h = b + \sum_{\forall i} x_i w_i$$

The activation function g allows the perceptron to perform nonlinear regression or classification, depending on the activation function used. Figure 9 shows four different activation functions which transform the h value (x axis) to some output value y. The *Identity*, *TanH* and *Logistic* activation functions perform regression and the *Binary step activation function* performs classification on the value of h.

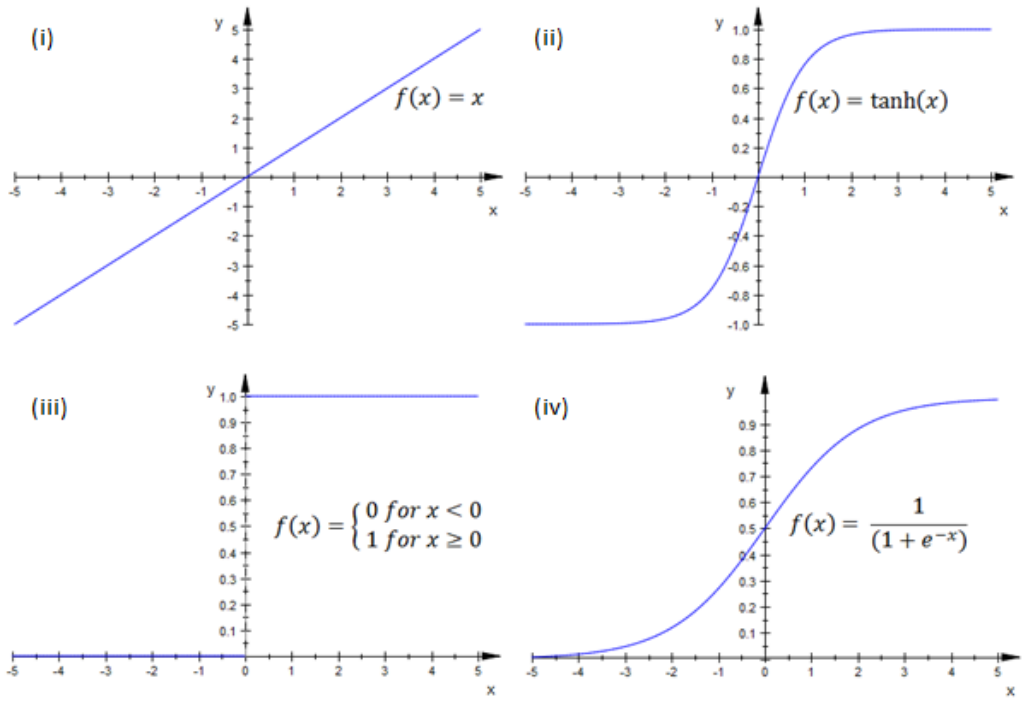


Figure 9: Four activation functions: (i) Identity; (ii) TanH; (iii) Binary step; (iv) Logistic

Regressions and classifications from multiple perceptrons can be combined to create a complex regression or a higher-level classification of regressions and classifications. These are called *multilayer perceptrons* or *artificial neural networks* and are discussed below.

2.4.1.2 Multilayer Perceptron Architecture

Multilayer perceptrons, also known as artificial neural networks, have the capacity to create sophisticated regression and classification by combining individual perceptron regressions and classifications. Except for input neurons, each neuron within an artificial neural network is itself a perceptron.

Artificial neural networks have three types of layers: an *input* layer, a *hidden* layer (which is itself usually multi-layer) and an *output* layer. The input layer contains a number of *input neurons* that introduce data into the neural network. The next layer(s) of a neural network are the hidden layer(s). Each hidden layer contains a number of *hidden neurons* that are similar or identical to the perceptron model discussed in the Single Layer Perceptron Section 2.4.1.1 above. The final layer of a neural network is the *output layer* which contains one or more *output neurons*. Output neurons also comprise perceptrons and deliver the final classification or regression output(s) of the neural network.

Figure 7 shows an example of a *feed forward* neural network. The directed arrows from each perception show the information flow between the neurons show information flowing left to right.

Deep learning is an emerging research topic within the artificial intelligence community which aims to find an efficient way to automatically extract features from data. *Convolutional neural networks* are feed-forward neural networks with many hidden layers to allow the neural network to learn and discover features. Recent research has also seen convolutional neural networks for deep learning (Mohamed, Dahl, & Hinton, 2012). Convolutional neural networks are computationally expensive due to the many hidden layers and neurons in the network. As computation becomes cheaper, more applications of deep learning will emerge from academic literature. It is probable that deep learning is already being used by hedge funds but details about their specific application would be vague or unknown due to the secretive nature of their operations.

Neural networks can learn the correct classification or regression depending on the features and data available, see 2.1. The neural network's weight parameters w_i can be adjusted to learn the correct classification or regression. For s*upervised learning*, an artificial neural network's weight parameters can be optimised by using a training dataset and a learning algorithm such as *backpropagation* (Eberhart & Shi, 2011), *Genetic Algorithms* (Hadavandi, Ghanbari, & Shavandi, 2010; Kuo, Chen, & Hwang, 2001) or *bacterial chemotaxis optimization* (Zhang & Wu, 2009) to optimise and learn the weights of the training dataset.

Neural networks can have many hidden neurons and hidden layers, increasing the number of weight parameters. This consequently increases the amount of processing power needed to learn the correct weights. Additional layers and weights can help the neural network learn the correct mapping between the input values and the output values but can increase the likelihood of overfitting, 2.3. Michael Azoff (1994) proposes a theorem that suggests the total number of hidden neurons is one plus twice the number of input neurons. However, he found the ideal number of neurons is dependent on the problem. Ultimately experimentation with different numbers of hidden neurons and hidden layers is needed to find the optimum number of layers and neurons for a specific problem.

2.4.1.3 Financial Application of Artificial Neural Networks

Various applications of artificial neural networks within the financial trading and investment literature have been explored. Artificial neural networks have been implemented to find optimal resource allocations for portfolios (Fernández & Gómez, 2007) and to make sense of how information obtained from the internet (from sources such as Twitter) affects the market (Bollen, Mao, & Zeng, 2011).

Forecasting future market movement is the most challenging application but has the potential to be very lucrative. Predicting market movement with artificial neural networks has also had some success. Chang, Liu, Lin, Fan and Ng (2009) used case-based reasoning with artificial neural networks to find undervalued stocks, classify the market (according to uptrend, steady and downward trend) and predict the highs and lows of the stock price. The authors achieved higher than 93.57%, 37.75% and 46.62% rates of return in stock markets.

Atsalakis and Valavanis (2009) offer a survey of soft computing methods that attempt to forecast the direction of stock markets. Over 100 papers were surveyed and 60% of the

papers used feed forward neural networks and recurrent neural networks. Input variables included technical analysis factors, fundamental analysis indicators, raw prices, changes between prices and lagged market data.

Beyond the use of standalone neural networks, ensembles of neural networks have been developed which are shown to outperform any individual neural network (Wang & Wu, 2011). Ticknor (2013) uses a Bayesian regularized network to help prevent overfitting by restricting the magnitude of the weights. Neural network approaches that employ other methods have also been developed. For example, Genetic Algorithms have been used to improve efficiency and forecasting ability by evolving the weights of a neural network to optimal values (Asadi, Hadavandi, Mehmanpazir, & Nakhostin, 2012).

Another variation on a standard neural network approach is a probabilistic neural network using a Bayesian method of classification (Chen, Leung, & Daouk, 2003). Claims by authors indicate that hybrid systems produce better results than pure neural network methodologies and the literature contains many such hybrid systems. See especially (Tang, 2009; Zhang G. , 2003; Das & Banerjee, 2011).

To capture as much market diversity as possible and reduce overfitting, authors suggest splitting an insample dataset into a training dataset containing the first 80% of market data, and a validation dataset of the remaining 20% to ensure that the neural network has not overfitted the training dataset (Zhang & Wu, 2009; Hadavandi, Ghanbari, & Shavandi, 2010; Fariaa, Albuquerquea, Gonzalez, Cavalcantea, & Albuquerquea, 2009). These are common settings amongst neural network practitioners.

In the literature, various metrics have been employed to measure how well the neural networks have learnt the objective using the sample data (Taylor, Darrah, & Moats, 2003). These metrics can be used to compare the trading system to other trading systems. Metrics include calculating the average profit and loss, the percentage of correct signal classifications and the number of signals the artificial neural network has produced.

Even though artificial neural networks are a “black box” approach to solving problems, they have been shown to be very successful when applied to asset price prediction (see for example (Atsalakis & Valavanis, 2009)).

No attempt has been made in literature to use artificial neural networks to classify whether traders, trading strategies or technical analysis algorithms are good or bad.

2.4.2 Support Vector Machines (SVMs)

Support vector machines were introduced by Vapnik (2013) and have been used to solve such problems as handwritten character recognition (Bahlmann, Haasdonk, & Burkhardt, 2002; Niu & Suen, 2012); protein classification (Fernandez-Lozano, et al., 2014); and image classification (Zhou, Wang, Zhang, & Wei, 2013). SVMs can be divided into linear support vector machines and nonlinear support vector machines which are discussed in subsections 2.4.2.1 and 2.4.2.2 respectively. The use of slack variables to handle the overlap between classes is discussed in 2.4.2.3. The application of SVMs in financial literature is discussed in 2.4.2.4.

2.4.2.1 Linear Support Vector Machines (LSVMs)

The goal of linear support vector machines is to design a hyperplane in data space that linearly classifies a training set of support vectors (data points in the training dataset) into two classes, whilst maximising the margin of the hyperplane (Cristianini & Taylor, 2000). For example, Figure 10 shows the relationship between two feature variables in the training dataset, '*Variable A*' and '*Variable B*' and two classes, '*Class 1*' and '*Class 2*'. The goal of the LSVM is to draw a line dividing the two classes whilst maximising the margin between the line and the support vectors.

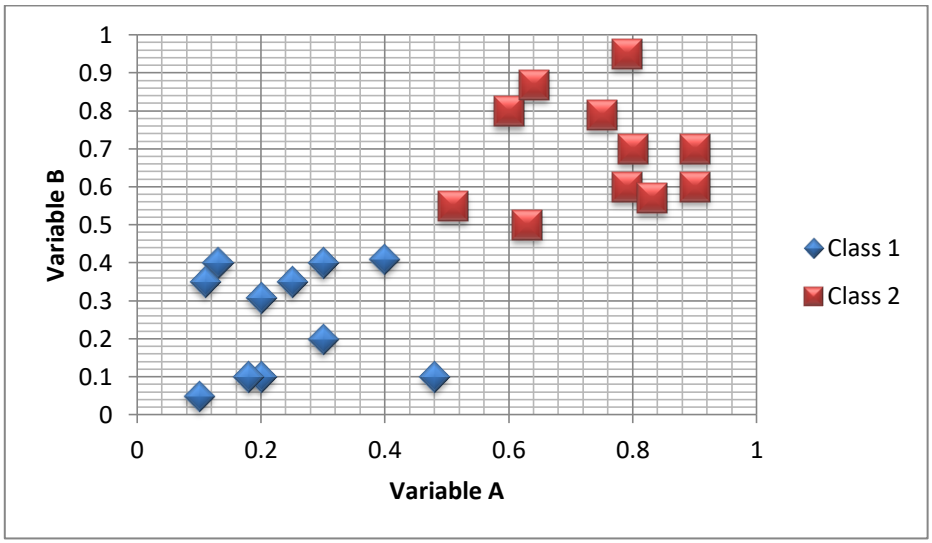


Figure 10: Graphical visualisation of two different classes

The two labelled classes can be separated by two different hyperplanes, as shown by the straight lines in Figure 11. The margin of a hyperplane as shown by the dashed lines in Figure 11 is the smallest distance between the hyperplane and the closest support vector perpendicular to the hyperplane. There are infinitely many hyperplanes that can divide the two classes shown in Figure 11, but the hyperplane that maximises the margin is globally optimal.

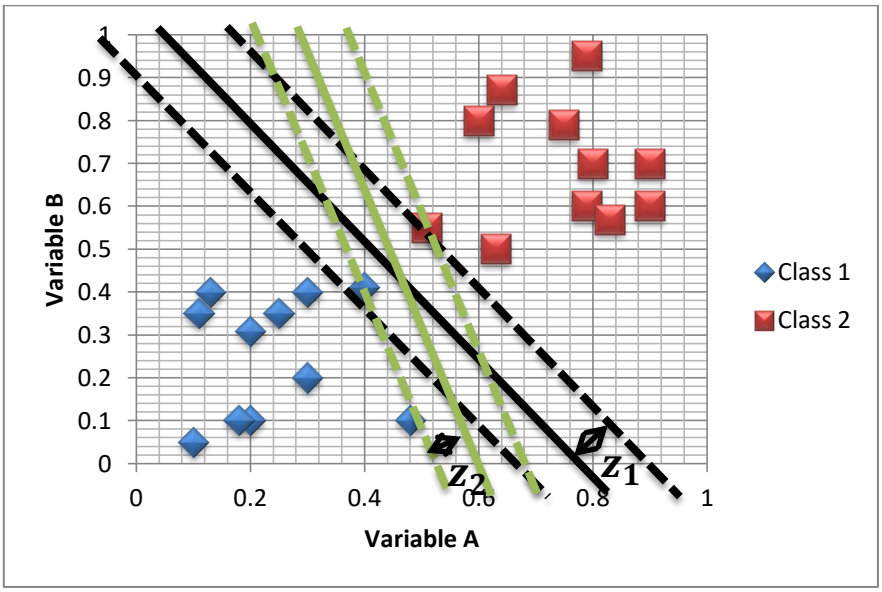


Figure 11: Two hyperplanes that separate both classes

A hyperplane is represented by the following equation,

$$w^T x + b = 0$$

where,

w^T is the normal vector,

b is the scalar value and

x is an input vector.

The normal vector determines the orientation of the hyperplane and the scalar value controls the hyperplane's displacement from the origin. The margins of a hyperplane, shown by two parallel hyperplanes, are given by the following equations,

$$w^T x + b = -1$$

$$w^T x + b = 1$$

The width of the margin, denoted as z_1 and z_2 in Figure 11, is equal to

$$\frac{2}{\|w\|}$$

Linear support vector machines are trained by maximising the width of the margin whilst obeying the constraint that the training data are correctly classified. This is an optimisation problem that can be solved by constructing a Lagrange function given the aforementioned constraints and objective function (Smola & Schölkopf, 2004)

Linear support vector machines classify these support vectors by using the following logic,

$$f(x) = \begin{cases} -1, & w^T x + b \leq -1 \\ 1, & w^T x + b \geq 1 \end{cases}$$

where -1 and 1 denote different classes.

2.4.2.2 Nonlinear support vector machines (NLSVMs)

Figure 12 shows a set of vectors which cannot be separated linearly by a hyperplane. To overcome this problem, practitioners use a 'kernel' function φ to nonlinearly map the vector's original feature space into a higher dimensional feature space to make the classes linearly separable by a hyperplane. The kernel function φ is applied before the linear support vector machine linearly separates the two classes in the new feature space. Commonly used kernel functions include the *Hyperbolic*, *Polynomial* and *Gaussian radial basis* kernel functions. Figure 13 shows an example of the new feature space after the kernel φ (below) is applied to the vectors in Figure 12.

$$\varphi\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{cases} \begin{pmatrix} 1 + x_1^2 \\ 1 + x_2^2 \end{pmatrix} & if\ x_1^2 + x_2^2 \geq 1 \\ \begin{pmatrix} x_1^2 \\ x_2^2 \end{pmatrix} & otherwise \end{cases}$$

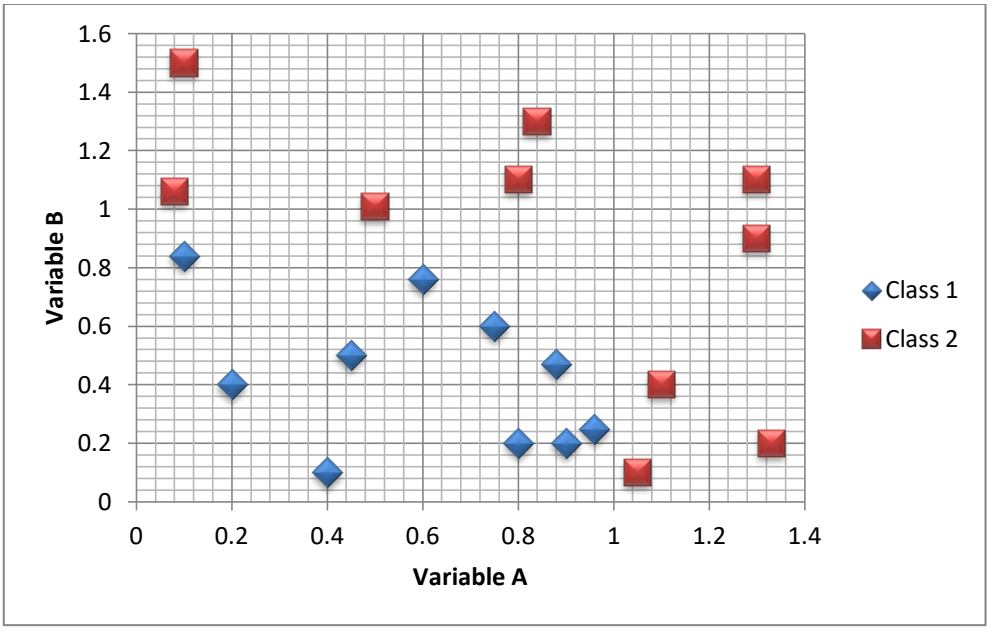


Figure 12: Nonlinearly separable classes

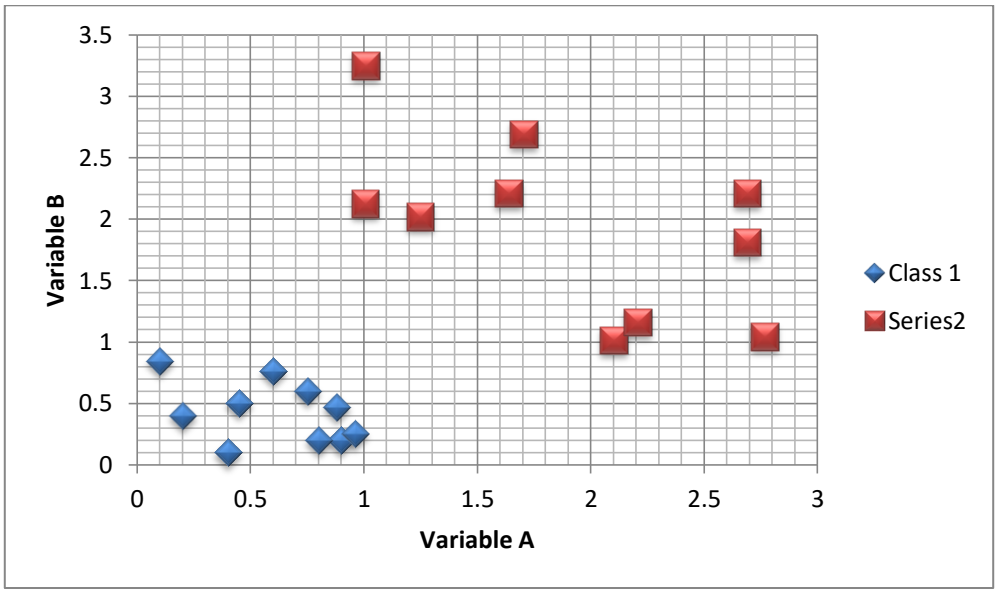


Figure 13: Linearly separable vectors mapped by the kernel function φ

2.4.2.3 Soft Margins

Figure 14 shows a training dataset of support vectors that are not linearly or nonlinearly separable. Overlapping classes, outliers and mislabelled vectors are considered noisy data. Practitioners may add more features in the hope that the classes become linearly or

nonlinearly separable (solvable with a kernel function) but for some real world problems, there is uncertainty in the actual value of the feature. Soft margins can be incorporated into the support vector machine algorithm by allowing support vectors to be misclassified, but to suffer a penalty in the objective function of the support vector machine algorithm (Diale, Walt, Celik, & Modupe, 2016). The soft margin approach removes the hard constraint that all vectors must be classified correctly, using instead a soft constraint (called a *slack variable* within the objective function) that allows some vectors to be misclassified (Tanveer, Shubham, Aldhaifallah, & Ho, 2016).

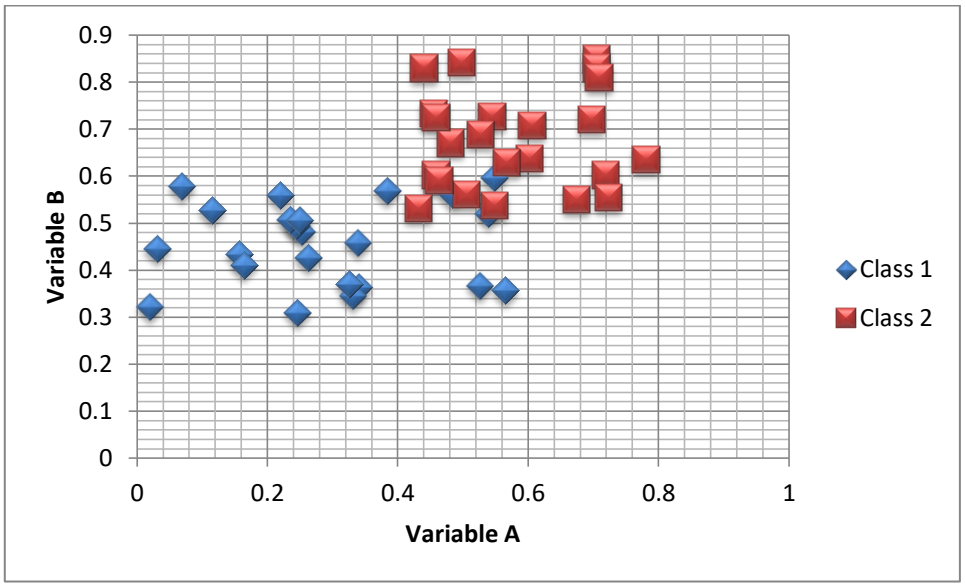


Figure 14: Feature overlap of two classes

2.4.2.4 Financial Application of Support Vector Machines

Gupta, Mehlawat, Inuiguchi and Chandra (2014) used support vector machines with a radial basis kernel function to select assets for a portfolio which suits a given investor-type. The predefined characteristics of an asset, short term return, long term return, risk and liquidity were used to classify the assets into three 'investor-type' classes. Support vector machines have also been used to predict the rate of return of a portfolio, with results compared to those of an artificial neural network's prediction (Hao, Wang, Xu, & Xiao, 2013); better portfolio selection was shown to be achieved using the support vector machine approach. Choudhury, Ghosh, Bhattacharya, Fernandes and Tiwari (2014) used an unsupervised learning technique called *K-means clustering* to cluster stocks into risk and return clusters.

Their hybrid system then used support vector machines to forecast the short term price movement and volatility to inform the general trading strategy of the portfolio.

Luss and D'Aspremont (2015) used support vector machines to predict abnormal returns using intraday price movements of financial assets by using text from news articles. They found that they could predict abnormal price movements but not anticipate the direction of the price movement. Similarly support vector machines have been used to classify press release documents into positive, negative or neutral documents from PRNewswire and Businesswire in order to determine the effect of the news on particular financial markets (Mittermayer, 2004).

Forecasting future market movements has met with some success. Kazem, Sharifi, Hussain, Saberi and Hussain (2013) describe a stock market price forecasting paper that uses a metaheuristic algorithm, the *Firefly algorithm*, to optimise the support vector machine. The firefly algorithm is a swarming algorithm that mimics the behaviour of fireflies. Fireflies are more attracted to other fireflies that are brighter and are more likely to move towards them, whereas fireflies that are dim are more likely to move randomly. The Firefly algorithm uses an objective function to determine the brightness of each firefly. The authors demonstrated that their novel system was more successful than other methods such as neural networks, a Genetic Algorithm optimised support vector regression and an adaptive neuro-fuzzy inference system for forecasting the market.

Papadimitriou, Gogas and Stathakis (2014) used support vector machines to provide short-term forecasting of the direction of daily average peak load. In this work, the only variables used were the volume and an autoregressive model. The authors commented that adding more variables may increase the support vector machine's forecasting success.

Kao, Chiu, Lu and Yang (2013) used a novel feature extraction which they termed *nonlinear independent component analysis*. The resultant independent components were used as features fed into the support vector regression algorithm. The authors compared their approach to other support vector regression systems. One of these did not use extracted features, another extracted the features from linear independent component analysis and another extracted features from principle component analysis. Using stock price data from the Nikkei 255 stock index and the Shanghai Stock Exchange Composite index, they reported

a lower prediction error with higher prediction accuracy when compared to other feature extraction techniques.

2.4.3 Ensemble Methods

Ensemble methods help to reduce the effect of overfitting which occurs when a classification or regression model memorizes the training dataset (described in 2.3). In ensemble methods multiple models are combined together to produce a better classification or regression model.

Consider this example; an ensemble classifier consisting of 3 weak classifiers can be used to outperform each individual weak classifier. Each weak classifier's misclassification can be shown to occupy some error space on a Venn diagram (Figure 15), in a manner suggested by Winston (2010) in which the Venn diagram constitutes a schematic illustration of the subdivision of the error space, i.e. the region of misclassifications. All 3 weak classifiers occupy a different error space. An unweighted 'majority vote' of these 3 weak classifiers correctly classifies all the data, as no two weak classifiers occupy the same error space.

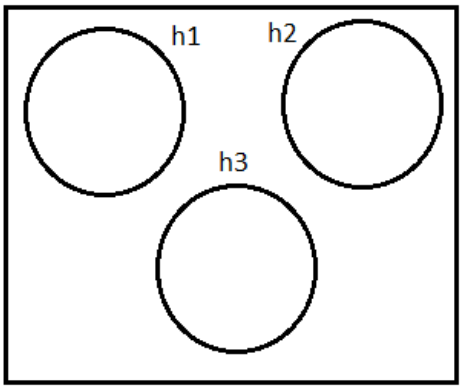


Figure 15: Venn diagram showing the error spaces occupied by weak classifiers, adapted from (Winston, 2010)

If parts of the weak classifiers' error spaces overlap (Figure 16) then the error of the ensemble of weak classifiers becomes the area of the intersection of the error spaces. If the intersected area is less than the error space of each weak classifier then the ensemble outperforms each of the weak classifiers individually.

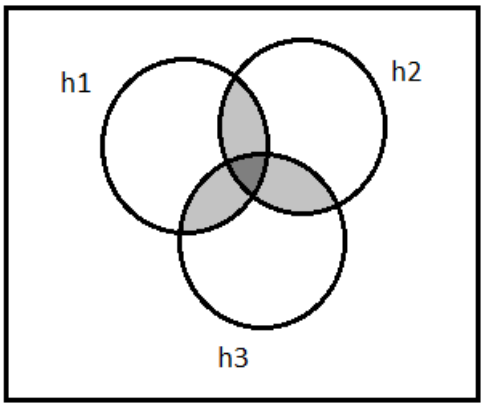


Figure 16: Venn diagram showing overlapping errors of weak classifiers, adapted from (Winston, 2010)

Extending this example further, consideration must be made of the selection of weak classifiers for the ensemble. Figure 17 shows the error spaces of weak classifiers that can be selected for the ensemble. An ensemble can be constructed such that no overlap of the error spaces occurs and that the majority vote ensures correct classification. Conversely, Figure 18 shows a badly constructed ensemble where error spaces overlap.

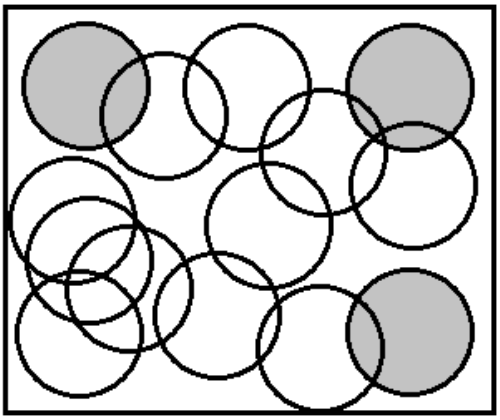


Figure 17: Venn diagram of an ensemble with no misclassification

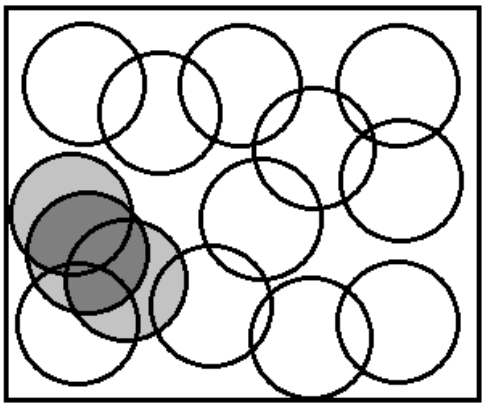


Figure 18: Venn diagram of a badly constructed ensemble

The classifier and/or regression models that are in the ensemble can be the same type of model or any combination of types of model, for example, artificial neural network, support vector machine or Adaboost classifier etc. Ensemble methods use a 'wisdom of the crowd' approach where the majority vote of a collection of predictions is better than any individual model's prediction.

The next section will outline two ensemble techniques that can help to reduce overfitting by combining classification and/or regression models together in 2.4.3.1. Then a popular ensemble method called Adaboost (described in 2.4.3.2) that will be heavily used in experiments later in this thesis, will be described in 2.4.3.2 before detailing financial trading applications of this machine learning algorithm in 2.4.3.3 and 2.4.3.4.

2.4.3.1 Bootstrap Aggregation and Feature Bagging

Bootstrap aggregation, also known as *bagging*, is an ensemble technique that combines classifier and/or regression models. This technique helps to reduce overfitting of high variance models with low bias in a model averaging technique. Each individual model is trained on a random subsample of data points (with replacement) from the training set. If the models are not severely overfitted, individual models should share common parts of the general classification which should emerge when individual models are combined. Models will memorize parts of their subsampled training set during overfitting. Overfitted regions of the model are less likely to occur when they are combined in a majority vote ensemble, as

each model is inclined to overfit on a different part of the training set and thus produce different error spaces.

*Feature bagging (*also known as the *Random subspace method*) is another ensemble method which helps to reduce overfitting of high variance models with low bias. It is possible that the classifier and/or regression models in an ensemble are too similar and that each model produces almost identical classifications or regressions. Feature bagging helps to solve this problem by training each model in the ensemble on a random subset of features, reducing the probability of constructing the same models.

2.4.3.2 Adaboost

Boosting is a supervised learning algorithm which uses ensemble techniques to create a classification or regression model. As with previously discussed ensemble methods, boosting is based upon the notion that the 'wisdom of the crowd' is better than the wisdom of any individual alone. At each iteration of a boosting algorithm, simple models are chosen which improve the classification or regression of the constructing model by correcting its errors.

Adaboost is a boosting algorithm which was first proposed in (Freund & Schapire, 1995). Adaboost has been successfully employed in handwritten character recognition (Lin, Song, Li, Wang, & He, 2017; Arth, Graz, Limberger, & Bischof, 2007), automatic speech recognition (Zhao, Xue, & Chen, 2015; Bergstra, Casagrande, Erhan, Eck, & Kégl, 2006) and object recognition (Cheng, Lee, & Guo, 2015; Tsai, Hsu, Chiu, & Chu, 2015; Opelt, Pinz, Fussenegger, & Auer, 2006). A notable object recognition algorithm designed for detecting faces featured the Viola-Jones algorithm (Viola & Jones, 2001) , and employed an Adaboost variant to detect objects from an image. This algorithm contained 180,000 simple classifiers. Each iteration of the boosting algorithm attempts to reduce the error of the strong classifier.

In the context of classification, the Adaboost algorithm constructs a classifier called a *strong classifier*, as a linear combination of weak classifiers of the form,

$$H = sign\left(\sum_{i=0}^{n} \propto_i h_i\right)$$

Where α_i is the weight, h_i is the weak classifier value, *H* is the strong classifier value, and the '$sign$' function indicates whether the summation is positive or negative.

Each weak classifier produces a 1 or -1 categorization output. Weak classifier output effectively determines whether to add or subtract the weight α from the total classification value. The higher the weight α, the more important the corresponding weak classifier is to the final classification decision. The strong classifier is a weighted collection of weak classifiers (unlike the example in 2.4.3 where each weak classifier contributed an equal vote to the final decision).

Each data point in the training set is associated with a weight value which represents how important the data point is to be classified correctly. Initially each data point in the training dataset is equally important so these values are all initialised to a normalised value of one divided by the number of data points in the training dataset.

A classification error value is calculated for each weak classifier by summing all of the weight values of the incorrectly classified data points in the training dataset. The weak classifier with the lowest error is chosen to be the best weak classifier. This weak classifier multiplied by the weight $\propto$ is added to the ensemble where the weight $\propto$ is calculated by the following formula,

$$\propto = \frac{1}{2}\ln\left(\frac{1 - error}{error}\right)$$

Next the weights of each row are updated by dividing the data points into two classification groups, *correctly classified* and *misclassified*, using our currently constructed ensemble.

The weight values of each group are then normalized to equal *0.5*. The new weights will place more importance on classifying the misclassifications of previous weak classifiers.

The process repeats until the machine learning practitioner decides to stop the Adaboost algorithm. This may occur when the strong classifier's error converges, validation data suggests the strong classifier is beginning to overfit, a certain amount of time has passed or the algorithm has iterated a certain amount of many times, etc.

2.4.3.3 Financial Applications of Boosting

This section will focus specifically on financial applications. Barinova and Gavrishchaka (2009) used boosting as an optimisation technique to allocate funds to different trading strategies rather than using boosting as a classification algorithm (Gavrishchaka V. V., 2006). Their 2009 paper used a regularization technique which removed noisy, hard-to-learn data points as the boosting algorithm iterated, as suggested in (Vezhnevets & Barinova, 2007). Not much has been done in applying adaptive boosting to portfolio asset allocation.

Salehi, Moradi and Molaei (2015) compares three algorithms, *least angle regression*, AdaBoost and *kernel ridge* regression to forecast systematic risk to help aid financial investment decisions. The authors derived 30 financial variables (fundamental metrics) from companies listed on the Tehran stock exchange and found the least angle regression outperformed Adaboost on its ability to forecast risk.

Creamer and Freund (2010) proposed a hybrid automated trading system. Their system used *LogitBoost*, a boosting algorithm which places AdaBoost into a statistical framework. Their system created alternating decision trees and combined them in a weighted majority vote, and the authors report positive returns and noted that Sell and hold would have done better or the same as trading with transaction costs.

2.4.3.4 AdaBoost Soft margins

The training dataset used by the Adaboost algorithm may contain errors. These errors could be outliers and/or mislabelled data points. Additionally the training dataset may be derived from a noisy data source and the features values may only be approximations which can create overlap between classes, see 2.4.2.3.

As Adaboost iterates, the strong classifier under construction gives more weight to data points which are misclassified, forcing the constructed strong classifier to overfit around these data points. While Adaboost rarely overfits in a low noise regime, it is not robust and

is prone to overfitting when used on noisy data because hard margin classification is suboptimal for the noise case. Intuitively, the dataset employed in the current project for trading strategy classification is necessarily noisy because trading strategy metrics are derived from market data, which is itself very noisy. Metrics derived from simulated market data hopefully reduce the noise but nevertheless noise still persists.

Several regularized methods and generalizations of the original Adaboost algorithm have been proposed to achieve soft margins. These include direct incorporation of a regularization term into the error function, and linear and quadratic programming with slack variables to improve existing ensembles (Rätsch, Onoda, & Müller, 2001).

BrownBoost is a variant of the Adaboost algorithm that assumes noisy data points repeatedly get misclassified and "gives up" on these data points as the algorithm iterates (Freund, 2001).

LPBoost is another variant which attempts to minimize misclassification error and maximize the soft margin (Demiriz, Bennett, & Shawe Taylor, 2002). This algorithm adjusts the weights of the strong classifier which is being constructed at each iteration of the algorithm.

Ada-GA is a proposed system which evolves a population of 'sub-classifiers' to reduce overfitting of the boosted strong classifier (Elden, Malaka A. Moustafa, & Emara, 2013). The 'sub-classifiers' are created by randomly selecting weak classifiers and their weights from the resultant strong classifier obtained from the Adaboost algorithm.

One algorithm which includes direct incorporation of a regularization term into the error function emplaces a low level of trust in data points which are highly weighted (Rätsch, Onoda, & Müller, 2001). A regularization term 'C' is included into the error function, and the higher the 'C' value the lower the trust in highly weighted data points. If 'C' equals zero then the algorithm behaves like the original Adaboost algorithm and for 'C' greater than 0 soft margins are introduced.

2.5 Evaluating Binary Classifiers

Binary classifiers attempt to correctly classify data as either one class or another, for example, good or bad, positive or negative. The performance of a binary classifier can be evaluated using formulas that are derived from a binary classifier's predicted classification

and the true classification of the data. Table 1 shows a contingency table of frequencies that are used to calculate the performance values for the binary classifier. The performance values of a binary classifier are summarized in Table 2.

	Actually good	Actually bad
Classifier predicted good	True positive (TP)	False positive (FP)
Classifier predicted bad	False negative (FN)	True negative (TN)

Table 1: Contingency table

	Formula	Description
Precision (P), Positive predictive value (PPV)	$\frac{TP}{TP + FP}$	The proportion of observations classified as positive that are actually positive
Negative predictive value (NPV)	$\frac{TN}{TN + FN}$	The proportion of observations classified as negative that are actually negative
Recall (R), True positive rate (TPR)	$\frac{TP}{TP + FN}$	The proportion of actually positive observations that are correctly classified as positive
Specificity (S), True negative rate (TNR)	$\frac{TN}{TN + FP}$	The proportion of actually negative observations that are correctly classified as negative
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	The proportion of observations that are correctly classified as positive or negative
F_1^P score (for true positives)	$\frac{2PR}{P + R}$	The harmonic mean of the precision and recall. Best at 1, worst at 0
F_1^N score (for true negatives)	$\frac{2S(NPV)}{NPV + S}$	The harmonic mean of the negative predictive value and specificity

Table 2: Binary classifier performance values

The performance of a binary classifier at classifying negative occurrences is given by the *negative predictive value* and the *specificity value*. The negative predictive value is the proportion of observations classified as negative that are actually negative. The specificity is the proportion of negative observations that have been correctly classified as negative. Figure 19 shows a Venn diagram which contain all the positive and negative observations; the space occupying the circle labelled 'A' denotes all the data points that are actually negative. The complement of 'A' contains all the data points that are actually positive. The space occupying the circle labelled 'B' denotes the circle of data points that the binary classifier's classified as negative. From this Venn diagram the formulas for the negative predictive value and specificity can be derived.

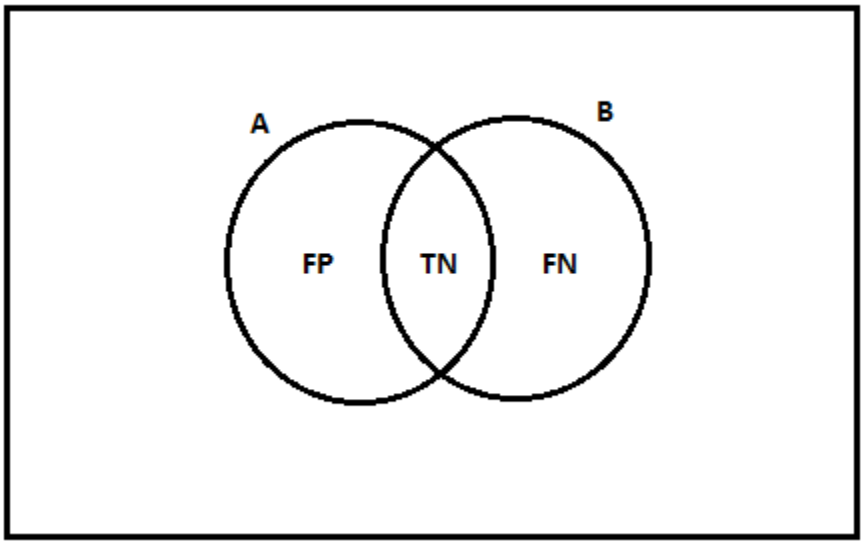


Figure 19: Venn diagram showing negative (A) and classified negative (B) data points

The performance of a binary classifier at classifying positive occurrences is given by the *precision* and *recall* values (Ting, Precision and Recall, 2010). The precision is the proportion of observations classified as positive that are actually positive. The recall is the proportion of positive observations that have been correctly classified as positive.

The accuracy performance measure (Ting, Accuracy, 2010) quantifies the overall performance of the classifier for classifying both the positive and negative data points.

Both the negative predictive value (Ting, Negative Predictive Value, 2010) and the specificity (Ting, Specificity, 2010) value are important when determining the performance of a binary classification system intended to identify and classify bad traders. The F_1^N score is the harmonic mean of the negative predictive value and the sensitivity, it is a single quantity that can be used to summarise just the identification and classification of bad traders. To help determine the binary classification system's performance at identifying good traders, the F_1^P score can be used (Ting, F1-Measure, 2010). The F_1^P score is a single quantity that can be used to summarise just the identification and classification of good traders. Binary classification systems can be compared by using any of the binary classification performance values.

2.6 Receiver Operating Characteristic (ROC curve)

The receiver operating characteristic curve (ROC curve) is a graphical plot of a regressor's recall and specificity (Bewick, Cheek, & Ball, 2004) (Hanley & McNeil, 1983). The ROC curve can also be calculated for classifiers if the classifier can predict a probability for its classification. Figure 20 shows the ROC curve of two classifiers. The ROC curve is created by ordering the probabilities or regression values from lowest to highest (where lowest denotes a classification of negative and highest denotes a classification of positive) and plotting the $Sensitivity$ against $1 - Specificity$ as the predicted probability (or regression) is varied from the lowest to the highest.

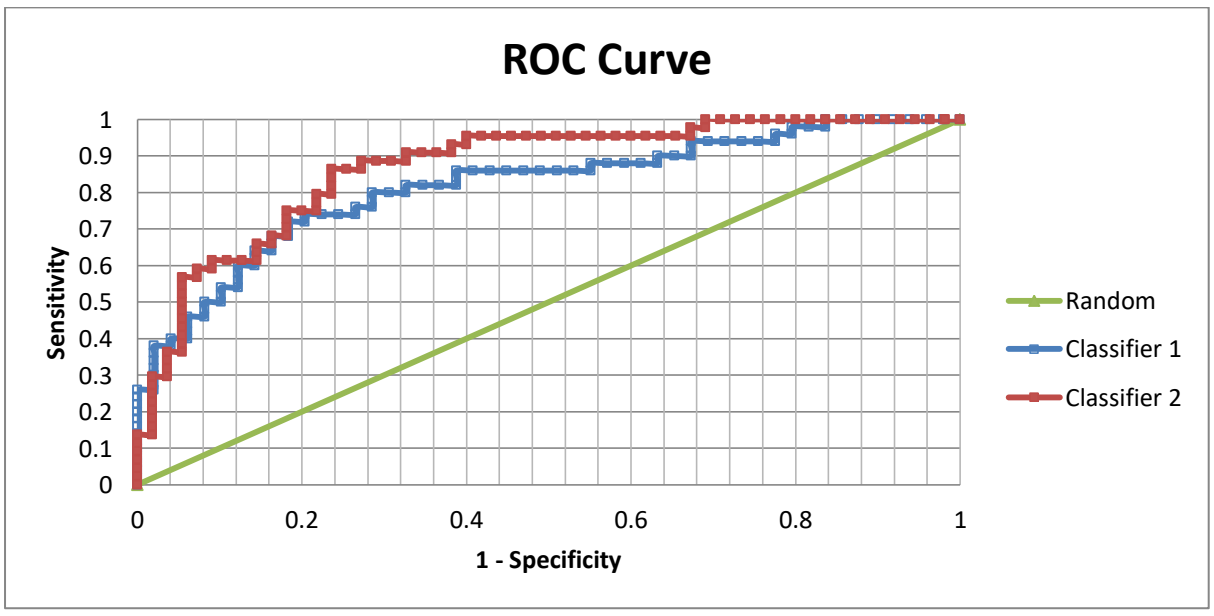


Figure 20: Receiver operating characteristic of two classifiers

ROC curves are used to visually compare classifiers by plotting the ROC curves of the classifiers on the same graph. The closer the ROC curve is to the diagonal line the closer the performance is to random. The area under a ROC curve (AUC) is an overall measure of performance. ROC curves that are furthest away from the diagonal have the largest such area. A randomly performing classifier has an AUC of 0.5 while a perfect classifier has an AUC of 1. In Figure 20, Classifier 1 has an AUC of 0.753 and classifier 2 has an AUC of 0.876. As classifier 2 has a larger AUC than classifier 1, the overall performance of classifier 2 is better than that of classifier 1. It can also be seen visually that classifier 2 is better than classifier 1 as classifier 2's ROC curve is furthest away from the random performing classifier.

2.7 Optimisation and Search

Local search is a method for solving computationally hard optimisation problems using heuristics (Hiep, Duc, & Trung, 2016). For some problems, the search space of possible solutions may be too vast for a brute force algorithm to be able to check every solution. Local search however does not check every possible solution but its goal is to heuristically search the search space and find a good enough solution in a reasonable amount of time. Local search algorithms such as Tabu Search and Simulated Annealing metaheuristic algorithms consider the current state of a solution then looks locally for better solution states using some objective function that can score solutions (Jackson, Özcan, & John, 2017). Genetic Algorithms utilise many solutions to find a local optimum solution. This is done by swapping a part of each solution with another solution at each iteration of the algorithm. Solutions are selected for swapping according to a fitness function (objective function), more detail in 2.7.1.

Consider the travelling salesperson problem; the salesperson has to visit x different cities, ideally using the fastest route possible. One approach to locating the fastest route from city to city is to search for it within the search space. However, it is impractical to search the search space by brute force if there are too many cities; for example, finding the global optimum solution with 20 cities creates 2,432,902,008,176,640,000 possible solutions.

As early as 1999, Mayer, Belward, Widell and Burrage (1999) examined Genetic Algorithms amongst several numerical optimization tools, and noted that these were faster in the

context of general systems models. Nicoara (2015) states, having compared Genetic Algorithms to several more traditional methods of optimization in the context of production decisions, that “a Genetic Algorithm is easy to implement, easy to extend and easy to parallelize in order to gain computational efficiency”. Hence Genetic Algorithms are a good choice for illustrating optimisation and local search, and will be used later in this thesis.

The main goal of this thesis is to classify bad traders, so this thesis will not cover every aspect of optimisation and local search in this review.

2.7.1 Genetic Algorithms

Genetic Algorithms were first formulised by Holland (1975). Genetic Algorithms were inspired by the way evolution works in the natural world, and provide a powerful framework which can be implemented to solve problems. As the Genetic Algorithm iterates, the population, converges towards a local optimum in the search space as defined by the Genetic Algorithm’s fitness function (Melanie, 1998). In many instances, local optimal solutions to problems are good enough and depending on the search space global optimal solution may be obtainable. The main advantage of this framework is that it enables the location of a good, but not necessarily the best, solution in a sufficiently short time to be useful.

Figure 21 outlines the Genetic Algorithm. The Genetic Algorithm has six main components which will be discussed in more depth in the following sections in this chapter:

1. Representation
2. Initialisation
3. Selection
4. Fitness function
5. Crossover
6. Mutation

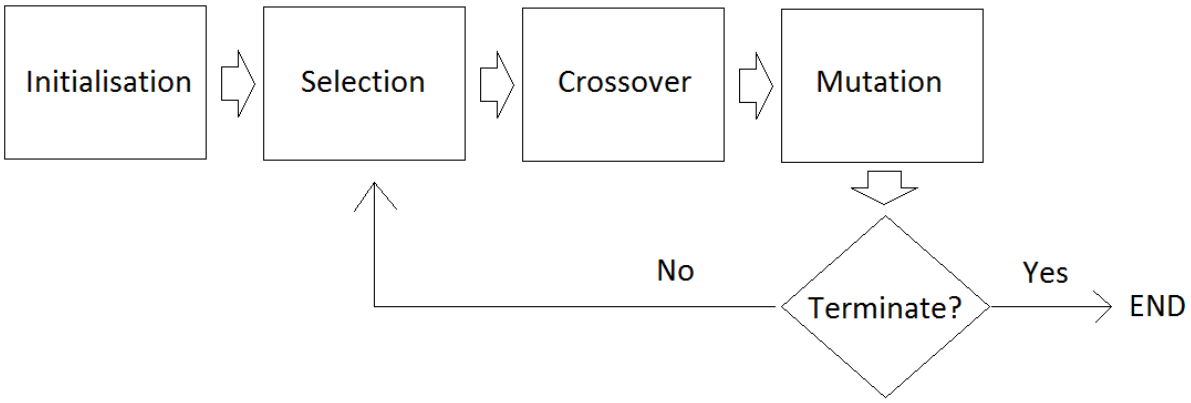


Figure 21: Genetic Algorithm outline

2.7.1.1 Representation

A solution to a problem is represented by a chromosome. A chromosome can be formulated many ways depending on the problem. For instance, the *travelling salesperson* problem is a combinatorial problem subject to *combinatorial explosion, i.e.* non-linear growth in the space of possible solutions given linear growth in items of interest. However, optimal solutions can be found by Genetic Algorithms (Braun, 1990; Whitley, Hains, & Howe, 2010). An example follows.

Consider the travelling salesperson problem outlined in 2.7. A Genetic Algorithm can find a suboptimal solution in a timely manner that may be sufficient. To construct a chromosome that represents a solution to this problem, all of the 20 cities are represented by a letter and each letter is placed into the chromosome (Figure 22). This is a one dimensional array representation in which each entry is referred to as a gene. The order of the cities within the chromosome denotes the path along which the traveling salesperson travels, starting from the first city in the first gene all the way through to the last city in the last gene.

Figure 22: Travelling salesperson chromosome

As another example of chromosome formulation, consider the problem of creating functions to fit a sequence of numbers. The functions can be represented by a tree-like chromosome (Figure 23). Each node represents a gene within the chromosome and can be replaced by other genes. It is clear that the chromosome represents the equation $10(12 + n)$.

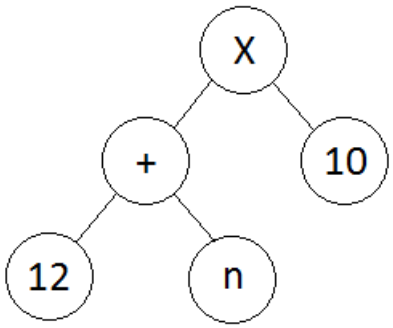


Figure 23: Tree-like chromosome

2.7.1.2 Initialisation

The goal of Genetic Algorithms is to evolve a population of chromosomes (solutions to some problem) to maximise some fitness function. The initialisation stage of the algorithm creates the initial pool of chromosomes. Usually the chromosome is given random gene values. For instance, a chromosome constructed as an array of zeros and ones can be initialised by randomly choosing either zero or one for each gene. More advanced initialisations can also be implemented such as the knowledge-based initialization technique published by Li, Chu, Chen and Xing (2015) which aims to seed the initial population with high quality chromosomes.

Care is needed when initialising some chromosome constructions. For instance, chromosomes in the travelling salesperson problem cannot have duplicate genes; this would result in the salesperson missing a city as the number of genes is equal to the number of cities to visit.

2.7.1.3 Selection - Stochastic Universal Sampling

Once the population is initialised, a fitness value for each chromosome is calculated. The fitness function that is used to calculate a chromosome's fitness is specific to the problem implemented. Consider a Genetic Algorithm implementation that attempts to predict the price direction of a company's future share price. A chromosome should obtain a higher fitness value if it can maximise profit and minimize risk.

Once the fitness values for all chromosomes are calculated, they are normalised so that each chromosome occupies a unique interval between zero and one (Figure 24). A random number between zero and one is generated which corresponds to an interval that is occupied by a chromosome. This chromosome is selected for the next stage of the Genetic Algorithm. As the bigger intervals are occupied by higher scoring chromosomes then the better chromosomes (solutions) with better genes are more likely to be selected. This selection process is done until the number of selected chromosomes equals the size of the population so that the next stage, the crossover stage of the algorithm, can mix the genes of the selected chromosomes to create the next generation of chromosomes.

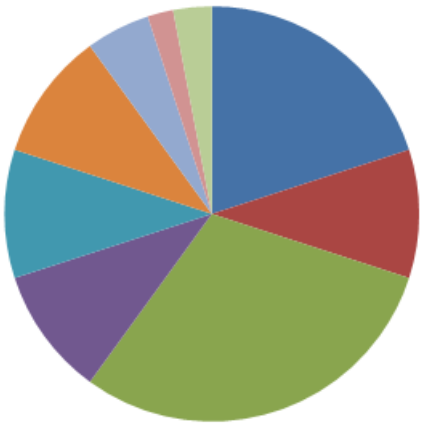


Figure 24: Pie chart representing chromosomes occupying intervals between 0 and 1

It is possible that a few chromosomes occupy a large proportion of the normalised range. Whilst the fittest chromosomes should be selected more often, it is important to select the

other chromosomes for the next stage of the algorithm as there needs to be diversity sets of genes to stop the population of chromosomes converging early. To overcome this, *stochastic universal sampling* can be employed. Stochastic universal sampling takes the generated random number and creates other numbers equally spaced apart. In four point stochastic universal sampling, a random number is generated and 3 other numbers which are spaced 0.25 away from each other and from the random number (Figure 25). Stochastic universal sampling therefore avoids individual chromosomes that occupy a large proportion of the normalized range. The four chromosomes corresponding to these numbers are selected for the next stage of the algorithm.

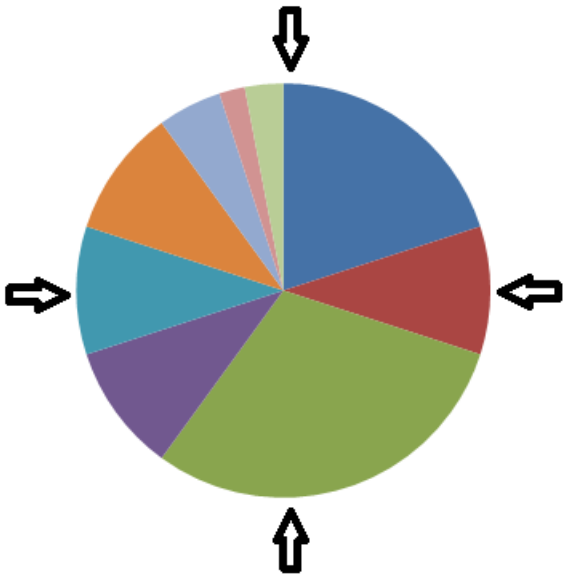


Figure 25: Four point stochastic universal sampling

The problem with the stochastic universal sampling selection technique is that negative fitness values cannot be normalised without transforming the fitness values in some way. In Chapter 6, a Genetic Algorithm is used to evolve a population of trading strategies by using a fitness function that maximises return. The fitness values produced are positive and negative, so to normalise these fitness values, all return values are shifted by the same number such that the trading strategy with the smallest return is above zero. This presents a scaling issue; the trading strategy with the highest return may be twice as profitable as the second most profitable trading strategy. After the shift for normalisation, the magnitude of the difference in profit is lost.

2.7.1.4 Selection – Tournament Selection

Tournament selection also selects the chromosomes for the next stage of the Genetic Algorithm. Random chromosomes are repeatedly selected to compete in small tournaments of size k in which the winner is selected for the crossover stage of the Genetic Algorithm.

The fitness values of each chromosome are sorted from the highest to lowest in the miniature tournament. For each chromosome from the highest to the lowest fitness value there is some probability p that the chromosome wins and is selected for the crossover stage of the Genetic Algorithm.

Unlike the previous method, the ranking of chromosomes means that negative fitness values do not need to be shifted for normalisation. Additionally chromosomes with overwhelmingly high fitness values are not at risk of being repeatedly selected, which can lead to the population of chromosomes to converging to one of little genetic diversity. On the other hand, chromosomes with the lowest fitness values become more likely to be chosen, and the Genetic Algorithm may take longer to converge on a good solution. However this can be counteracted by increasing the tournament size so that weak chromosomes have a smaller chance of being selected.

2.7.1.5 Crossover

During the crossover stage, two chromosomes are chosen at a time in a process referred to as 'crossover' (analogous to 'breeding'). The two chromosomes have a probability p that they swap genetic material and their offspring are in the next stage of the Genetic Algorithm. The two chromosomes have a probability of $1 - p$ that the two chromosomes do not swap genetic material and are copied to the next stage of the Genetic Algorithm.

Two chromosomes can be crossed over in many different ways, depending on the construction of the chromosome. Some crossover algorithms may not be suitable because they do not produce a valid chromosome. For example, swapping genetic material of two travelling salesperson solutions (chromosomes) may result in solutions that do not contain all the cities and have duplicate cities in the solution.

One point crossover is a simple crossover operation on two chromosomes containing an array of *n* genes. A random value between 1 and *n*-1 is generated which denotes the index

between the genes. At this index, the two chromosomes exchange their genes past that index, and then these new chromosomes enter the mutation stage. In Figure 26 the random number is 4 so all the genes past the 4th gene are exchanged.

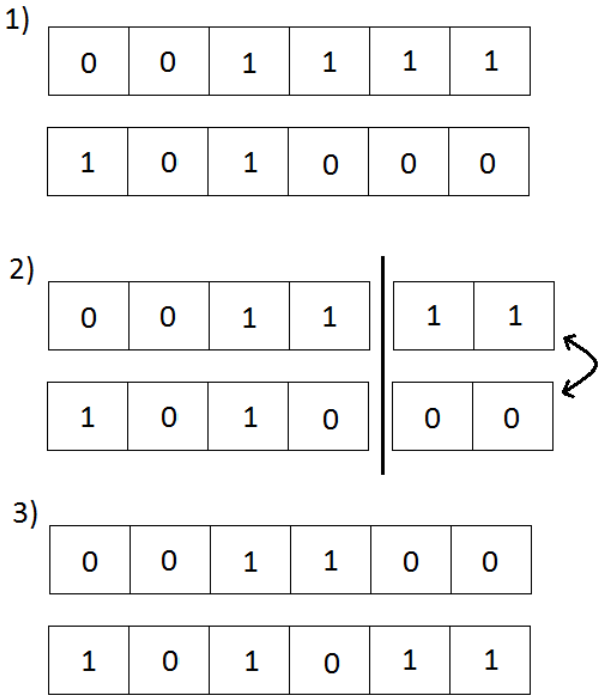


Figure 26: Single point crossover between two binary chromosomes

However, the use of one point crossover in the travelling salesperson problem (for example) would create an invalid chromosome. Thus other crossover methods like the *permutation crossover* are used. The permutation crossover will ensure there are no duplicates within each chromosome.

2.7.1.6 Mutation and Termination Conditions

At this stage of the algorithm, genes within the chromosomes can mutate. This introduces new genes into the population which may not have existed during the initialisation process or have become extinct by previous iterations of the Genetic Algorithm. By altering the genes it becomes more likely that the population will contain chromosomes capable of escaping local optima to arrive at new solutions. To put this in another way, it is possible that genes that were not previously needed to improve on a current best solution might, at some point in the future, prove necessary to escape some local optimum.

Mutation can be implemented in many different ways, depending on the chromosome's construction. For chromosomes which are given as an array of binary digits (Figure 26), each gene might have, say, a 0.1% chance of mutating to the other digit.

After mutation has occurred the population can be evaluated according to the same fitness function used during the selection process. The fittest chromosome in the population is compared to the fittest chromosome so far, and this fittest chromosome is recorded. Then the Genetic Algorithm from the selection stage is repeated, until some termination condition is fulfilled.

There are a wide range of termination conditions that can be implemented in the Genetic Algorithm:

- The population has converged to a single (or a few) chromosome(s) duplicated throughout the population;
- Sufficient time has passed;
- Sufficient iterations of the algorithm have taken place;
- The best chromosome so far has exceeded some measure; or
- The best chromosome has not improved over the last *n* iterations.

2.7.1.7 Genetic Algorithm Literature on Creating Trading Strategies

In the literature there have been many Genetic Algorithm implementations which aim to combine variables and/or models into trading strategies or trading rules. Lakshman, Ramesh, Manjula and Govardhan (2012) proposed a Genetic Algorithm which evolved a population of 100 trading rules using the return performance metric on historical data as the fitness function. Each trading rule consisted of 6 conditions which are technical analysis algorithms that have a threshold above or below a particular value. If all conditions are satisfied, then the trading rule produces a buy signal else the trading rule produces a sell signal. The authors used the stock prices from the India Cements stock index future. A limited amount of market data of 80 trading days from September 2011 to December 2011 were used as historical data and 80 trading days for from January 2012 to April 2012 were used as outsample data. Only six trading rules were evolved and they were compared to

the buy and hold strategy. The trading rules reported 21% - 54% return on outsample data compared to -26% for the buy and hold strategy.

Lipinski (2010) has represented trading experts by a binary chromosome in which each gene indexes a trading rule. If a gene value is 0 then the trading rule is not active and if the gene value is 1 the trading rule is active. The trading experts predict market price direction by taking the majority vote of all buy and sell signals from the trading rules active. Lipinski has also created trading experts (chromosomes) for which the gene contains a number representing the weight a trading rule has on the final decision of the trading expert (Lipinski, 2008). Lipinski has explored the use of objective functions such as the Sharpe ratio, Sortino ratio, Sterling ratio and the Treynor ratio, performance metrics of return and risk (Lipinski & Korczak, 2004). The Sharpe ratio, Sortino ratio and other performance metrics are explained in Chapter 4 Lipinski has also experimented with fixing common building blocks of trading rules and then running the evolutionary process, which is reported to reduce the search space (Lipinski, 2010).

Recently, Ozturk, Toroslu and Fidan (2016) published a paper that uses Genetic Algorithms to evolve binary chromosomes in a similar fashion to Lipinski's work. Their hybrid system also used the Genetic Algorithm to optimise the parameter settings of a population of trading rules. Each trading strategy contains 20 trading rules and the binary gene values activate or deactivate the trading rule. The trading strategy makes buy and sell decisions based upon the majority vote of all the activated trading rules. The authors performed 3 experiments, the first used 1 minute EURUSD data from the start of January 2013 to the end of December 2013. In total, 246,792 data points were used as historical data and 121,648 data points for outsample. The second experiment used 1 minute EURUSD from the start of January 2013 to the end of June 2013. In total, 121,129 data points were used as historical data and 61,371 data points for outsample. The third experiment used 1 minute GBPUSD from the start of January 2014 to the end of June 2014. In total, 121,196 data points for historical data and 61,068 data points for outsample. The authors experimented with the return performance metric and the average profit per trade performance metric as fitness functions and found the return performance metric to be preferable; these performance metrics are explained in more detail in Chapter 4. About 60% of the optimised trading strategy trades were reported to be profitable.

Lipinski (2007) created trading experts in the form of multi-layered perceptrons, each having 15 trading rules as inputs. He then extended his work by creating trading experts (chromosomes) that contain artificial neural networks and trading rules (genes) (Lipinski, 2008).

Other literature uses Genetic Algorithms to optimise the parameters of some trading strategy template. Wang, An, Xia, Liu, Sun and Huang (2014; 2016) used Genetic Algorithms to optimise the parameters of two moving averages (moving average technical analysis algorithms are explained in 3.1). A highly cited paper authored by Allen and Karjalainen (1999) created trading strategies using mathematical operators, conditional operations, moving averages results and prices to create functions that output buy and sell signals.

Hu, Liu, Zhang, Su, Ngai and Liu (2015) observed a significant bias in their literature review of the discovery of trading rules in algorithmic trading on stock markets towards the application of Genetic Algorithms. The review found that the return performance metric (and measures that are essentially return quantities) were mainly used. The joint second most used performance measure was the Sharpe ratio performance metric and the root mean squared error.

Genetic Algorithms have been used in hybrid approaches evolving support vector machines and creating their own system of models (Yu, Wang, & Lai, 2005; Chiu & Chen, 2009). Chiu and Chen achieved about 71%-75% return on their capital in their model which combined Genetic Algorithms, artificial neural network and support vector machine methodologies. The Genetic Algorithm in Chiu and Chen's paper optimised the parameters of the fuzzy models. The fuzzy models are based on technical indicators and macroeconomic variables.

Modern Portfolio Theory helps investors construct a portfolio that maximises expected return given an investor's preference to risk. To construct the efficiency frontier which contains the optimal allocations of capital for various risk preferences, the following information is required: the investible assets, the correlation between the assets returns, each asset's expected return and the standard deviation of each asset's returns (which is a proxy for risk). Similarly Genetic Algorithms have been used to optimise the allocation of capital of a portfolio for long term investments (Qu, Zhou, Xiao, Liang, & Suganthan, 2017). In Modern Portfolio Theory the advantage of using local search such as Genetic Algorithms

to search for optimal portfolio allocation over a brute force method is that the Genetic Algorithm's runtime scales better as more assets are considered. Another implementation of Genetic Algorithms involves evolving a population of portfolio allocations where each gene in a chromosome gene denotes the proportion of capital to allocate to a particular asset (Krzysztof, Filipiak, & Lipiński, 2012; Aranha & Hitoshi, 2009). The Sharpe ratio performance metric can be used as a fitness function that would guide the population to a state that minimises risk and maximises return.

2.8 Conclusion and Direction of the Work

The application of Genetic Algorithms to trading strategy creation demonstrates two perspectives in published work. One focuses on the derivation of formulas and the optimisation of parameters of trading strategies; this perspective is the most prevalent in published work. The other uses Genetic Algorithms to combine technical analysis algorithms to create trading strategies.

In the literature, forecasting systems using artificial neural networks and support vector machines use technical analysis algorithms as inputs. This takes a 'low-level' approach to price prediction which seeks to make sense of technical analysis interpretations. However, this model of a trading strategy is a 'black box'. This approach seems unhelpful for this project, given the involvement of the industrial partner and the associated practitioner outlook. The parameters of the technical analysis algorithms are fixed and only a small number of technical analysis algorithms are used. The basic principle of combining technical analysis algorithms to form profitable models is in line with the direction of the project of this thesis.

Evolutionary approaches offer the prospect of adapting to changing market conditions (although the period of market data used for training is important here, and atypical changes in market behaviour are likely to remain problematic for market prediction). To date, only the work of Lipinski and the newly published paper authored by Wang, An, Xia, Liu, Sun and Huang, combines technical analysis algorithms into an evolutionary approach. By switching algorithms on or off within the chromosome, the pattern of those switched on could be thought of as constituting a trading strategy. However, the Genetic Algorithm

method employed does not control the numbers of trading rules that are active within these trading strategies, and so no explicit modelling of traders is evident.

Artificial neural networks and support vector machines are ‘black boxed’ approaches to regression and classification. The goal of this thesis is to create an early warning detection system for bad traders. The project partner has expressed that they would like to understand the decisions of the classification system which classifies bad traders. As a result, an Adaboost based classification system will be implemented and experiments using ensemble methods to create more robust classification systems will be explored.

In literature, no attempts have been made to classify whether a trader is good or bad using the machine learning classification or regression algorithms described in this thesis. The work described in this thesis aims at more explicit modelling of trading strategies.

Chapter 3 – Overview of Technical Analysis Algorithms

There are many technical analysis indicators and categories of these. Traders will form their own opinions as to which are most appropriate, or reliable, for their own trading decisions. The algorithms selected for experimentation in this thesis are all in common usage and are favoured by the industrial partner, as identified by the author's own induction training at that company. Technical analysis algorithms were chosen instead of fundamental analysis due to the ease with which they can be implemented in code, and the ability to easily derive necessary information from market data.

As described in Section 1.3.3, technical analysis employs primarily the price and volume data of an asset to predict future price movement. Unfortunately there are many ways in which to interpret these models, and in general there will even be conflicting interpretations (Lo, Mamaysky, & Wang, 2000). The technical analysis algorithms used in this thesis are the following:

- Simple Moving Average (Section 3.1.1)
- Simple Moving Median (Section 3.1.2)
- Exponential Moving Average (Section 3.1.3)
- Moving Average Convergence Divergence (Section 3.1.5)
- Bollinger Bands (Section 3.2.1)
- Relative Strength Index (Section 3.3.1)
- Stochastic Oscillator (Section 3.3.2)

A *technical analysis interpretation* is a subjective interpretation of a given technical analysis algorithm that instructs a trader on a course of action. For example, a condition might be used in combination with a technical analysis algorithm, with an output meeting the condition leading to the interpretation of the output

3.1 Trend Technical Analysis

Trend is the general direction of the market's price and is modelled by moving average algorithms. Moving average algorithms are used to smooth noisy candlestick data which help market participants identify trends and market cycles.

3.1.1 Simple Moving Average (SMA)

The simple moving average (Figure 27) is calculated by averaging the last n close prices, using the following formula (Dash & Dash, 2016),

$$SMA_i(n) = \frac{\sum_{j=0}^{n-1} C_{i-j}}{n}$$

where i denotes the i^{th} candlestick, j denotes the offset from the i^{th} candlestick, n is the number of historic close prices, and C_a is the close price at the a^{th} candlestick.

Simple moving averages with smaller n values respond to price changes more quickly (Figure 27) than those with large n (Figure 28) because they are averaging over fewer close prices.

Figure 27: Candlestick chart of hourly AUDUSD data with a simple moving average (in red) that averages the last 24 close prices

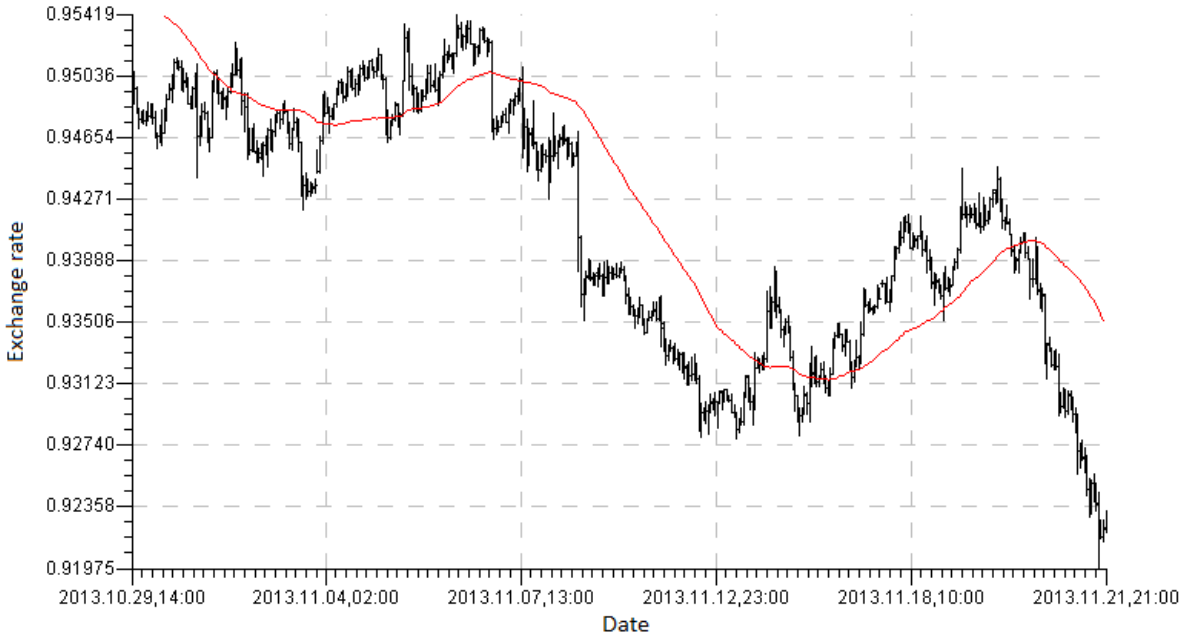


Figure 28: Candlestick chart of hourly AUDUSD data with a simple moving average (in red) that averages the last 60 close prices

3.1.2 Simple Moving Median (SMM)

Similar to the simple moving average algorithm that averages over the last n close prices, the simple moving median (Figure 29) is calculated by finding the median of the last n close prices.

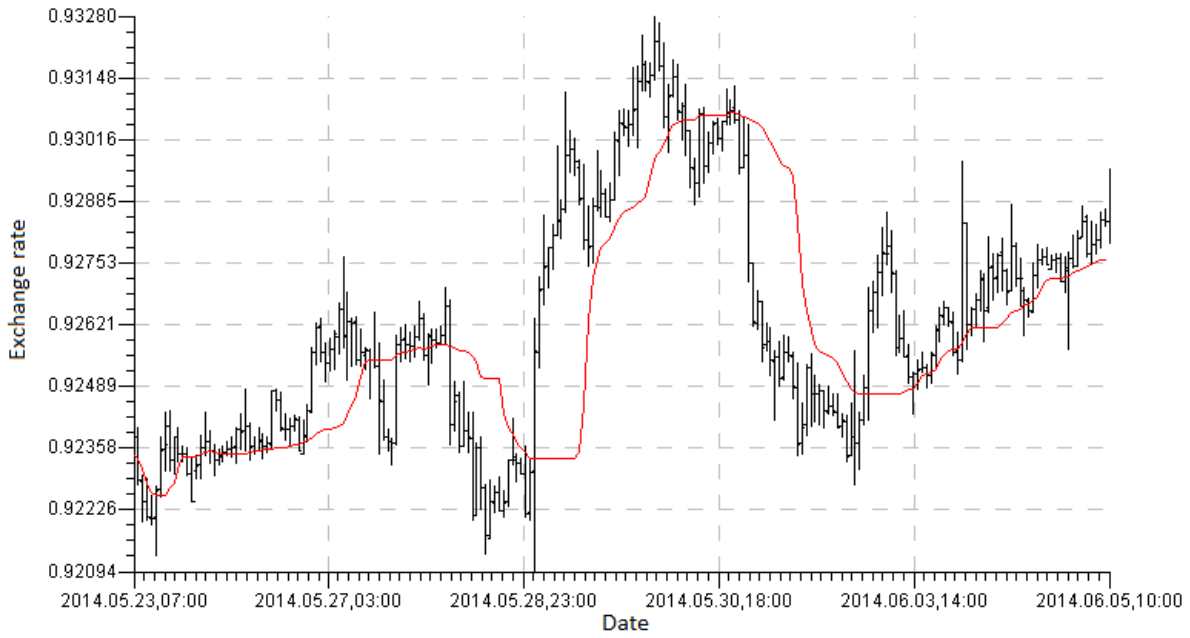


Figure 29: Candlestick chart of hourly AUDUSD data with a simple moving median (in red) that is the median close price of the previous 60 close prices

3.1.3 Exponential Moving Average (EMA)

The exponential moving average (Figure 30) assigns a greater weight the more recent the close price. EMA has the following formula (Macedo, Godinho, & Alves, 2017),

$$EMA_i(n) = EMA_{i-1}(n) + \propto (C_i - EMA_{i-1}(n))$$

$$\propto = \frac{2}{n-1}$$

where i denotes the ith candlestick, $\propto$ denotes the smoothing factor, n is the number of historic close prices, and C_a is the close price at the a^{th} candlestick.

Figure 30: Candlestick chart of hourly AUDUSD data with an exponential moving average (in red) using the previous 60 close prices

The smoothing factor $\propto$ is a constant coefficient between 0 and 1 that represents the degree of weighting decrease. The larger the smoothing factor value, the less the older close prices exert an influence on the total average.

3.1.4 Trend Interpretations

The output of technical analysis algorithms that model trend can be interpreted in various ways when reaching a buy/sell decision. In implementing technical analysis algorithms individually, or in combination, in this thesis to reach such decisions an exhaustive approach is taken to implementing algorithms and interpretations together. For the output of trend algorithms (simple moving average, simple moving median and the exponential moving average) the following interpretations are considered:

1. Uptrend if the current price is above the moving average.
2. Uptrend if the moving average previously is below the current moving average.
3. Uptrend if the price is above the last x moving averages.
4. Uptrend if the moving average value is above the last x moving averages.
5. Uptrend if the price is above the average of the last x moving averages.

6. Uptrend if the moving average value is above the average of the last x moving averages.

Downtrends are instead considered if conditions are reversed.

3.1.5 Moving Average Convergence/Divergence (MACD)

The moving average convergence/divergence algorithm (Figure 31) uses the difference between the short term and long term trends to anticipate future movements, as shown by the following formula (Ye, Zhang, Zhang, Fujita, & Gong, 2016),

$$MACD(N, n) = EMA(N) - EMA(n)$$

where n is the number of historic close prices for the shorter term trend, N is the number of historic close prices for the longer term trend, and $\mathrm{N} > n$.

Secondly, a moving average of the $MACD$ called the *signal line* (coloured blue in Figure 31) can be overlaid on the $MACD$ (the red line) to help traders identify trend reversal. A trend reversal is where a trend changes from an upward trend to a downward trend or vice versa.

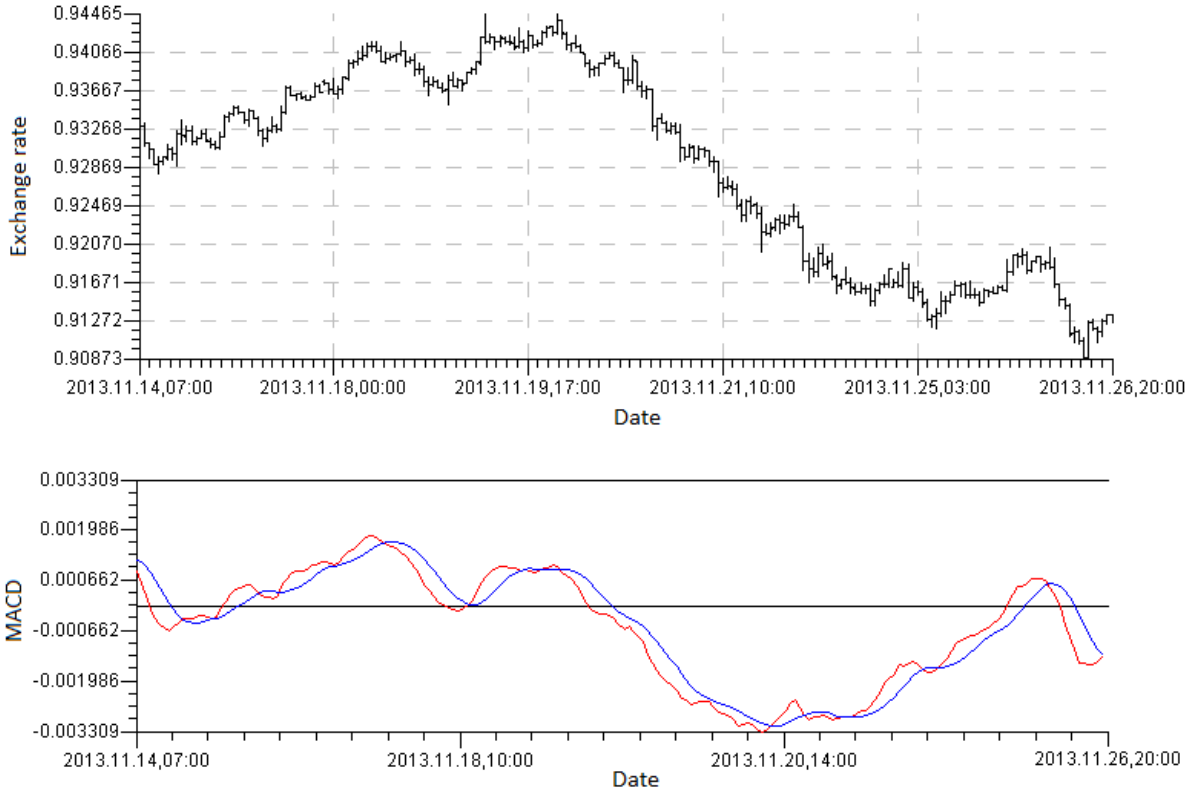


Figure 31: Candlestick chart of hourly AUDUSD data with a MACD indicator below. The MACD(26, 12) is shown in red and the signal line, shown in blue, averages the previous 9 MACD values

3.1.6 MACD Interpretations

The moving average convergence divergence algorithm can be interpreted in various ways. The interpretations implemented in this thesis are as follows:

1. Buying momentum (market conditions favouring profitable buying) if the MACD crosses the signal line from below to above the signal line.
2. Buying momentum if the MACD crosses the zero value from below to above the zero line.

Selling momentum instead considered if conditions are reversed.

3.2 Volatility Technical Analysis

Volatility is a measure of risk. Statistical measures such as the variance and standard deviation of the price series are often used to measure volatility. *Non-volatile* markets

experience steady price movements whilst *volatile* markets experience rapid upward and downward price movements over a short period of time.

3.2.1 Bollinger Bands (BB)

Bollinger bands (illustrated in Figure 32) were created by John Bollinger in the 1980s as a method for market participants to identify extreme short-term price movements in a security. Bollinger bands model the volatility of the current price action given the last n close prices. The upper and lower bands are created by calculating the standard deviation σ of the last n close prices from a moving average (Bollinger, 2001):

$$BB(n) = \begin{cases} SMA(n) + \sigma \\ SMA(n) \\ SMA(n) - \sigma \end{cases}$$

where σ denotes the standard deviation of the close prices, i denotes the i^{th} candlestick, j denotes the offset from the i^{th} candlestick, n is the number of historic close prices, and C_a is the close price at the a^{th} candlestick.

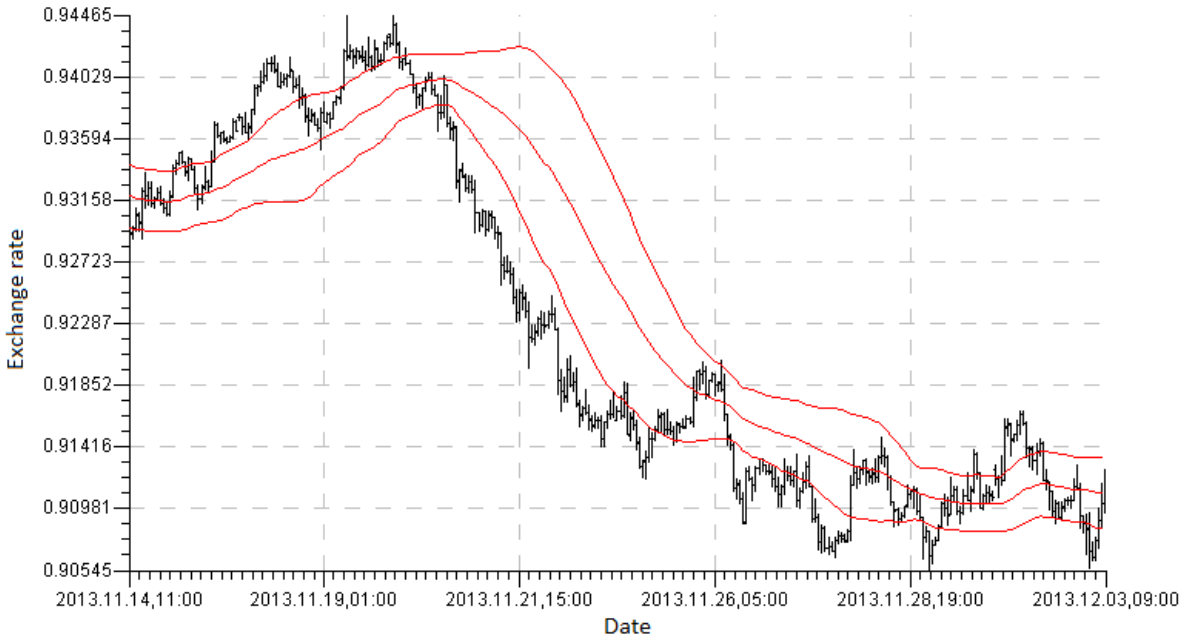


Figure 32: Candlestick chart of hourly AUDUSD data with a Bollinger band using the last 60 close prices. The middle red line is SMA, the higher red line is the upper band and the lower red line is the lower band, both bands are one standard deviation from the SMA

3.2.2 Bollinger Band Interpretations

The output of the Bollinger band algorithm can be interpreted in various ways. Here are the interpretations implemented in this thesis:

1. Buying momentum if the price is greater than the upper band. Selling momentum if price is lower than the lower band.
2. Overbought if the price hits the upper band so downward momentum once price crosses the upper band from above. Oversold if the price hits the lower band so upward momentum once price crosses the lower band from below.
3. Overbought if the price hits the upper band so downward momentum once price crosses the moving average after being above the upper band. Oversold if the price hits the lower band so upward momentum once price crosses the moving average after being below the lower band

3.3 Momentum Technical Analysis

Momentum models model the buying and selling pressures of market participants and give an indication of the strength. Momentum helps determine whether the market is *overbought* (the price has risen unjustifiably on the strength of high-volume buying), or *oversold* (the price has fallen unjustifiably on the strength of high-volume selling).

3.3.1 Relative Strength Index (RSI)

The relative strength index (illustrated in Figure 33) is a momentum indictor that helps determine whether the market is overbought or oversold given the last n close prices (Bell, 2016). The relative strength index uses a ratio that takes the last n price changes (the candlestick's close price minus the same candlestick's open price) and divides the average price increase by the average price decrease of the past n price changes. The indictor is calculated by the following formulas:

$$RSI(n) = 100 - \left(\frac{100}{(1+\alpha)}\right)$$

$$\alpha = \frac{Avg_{up}}{Avg_{down}}$$

where, Avg_{up} is the average price of increasing candlesticks and Avg_{down} is the average price of decreasing candlesticks.

When the RSI value is above 70, this indicates that there is buying momentum and the market is overbought. If the RSI value is below 30 then there is selling momentum and the market is oversold. The values 70 and 30 are typical thresholds and are normally the default settings on most trading platforms.

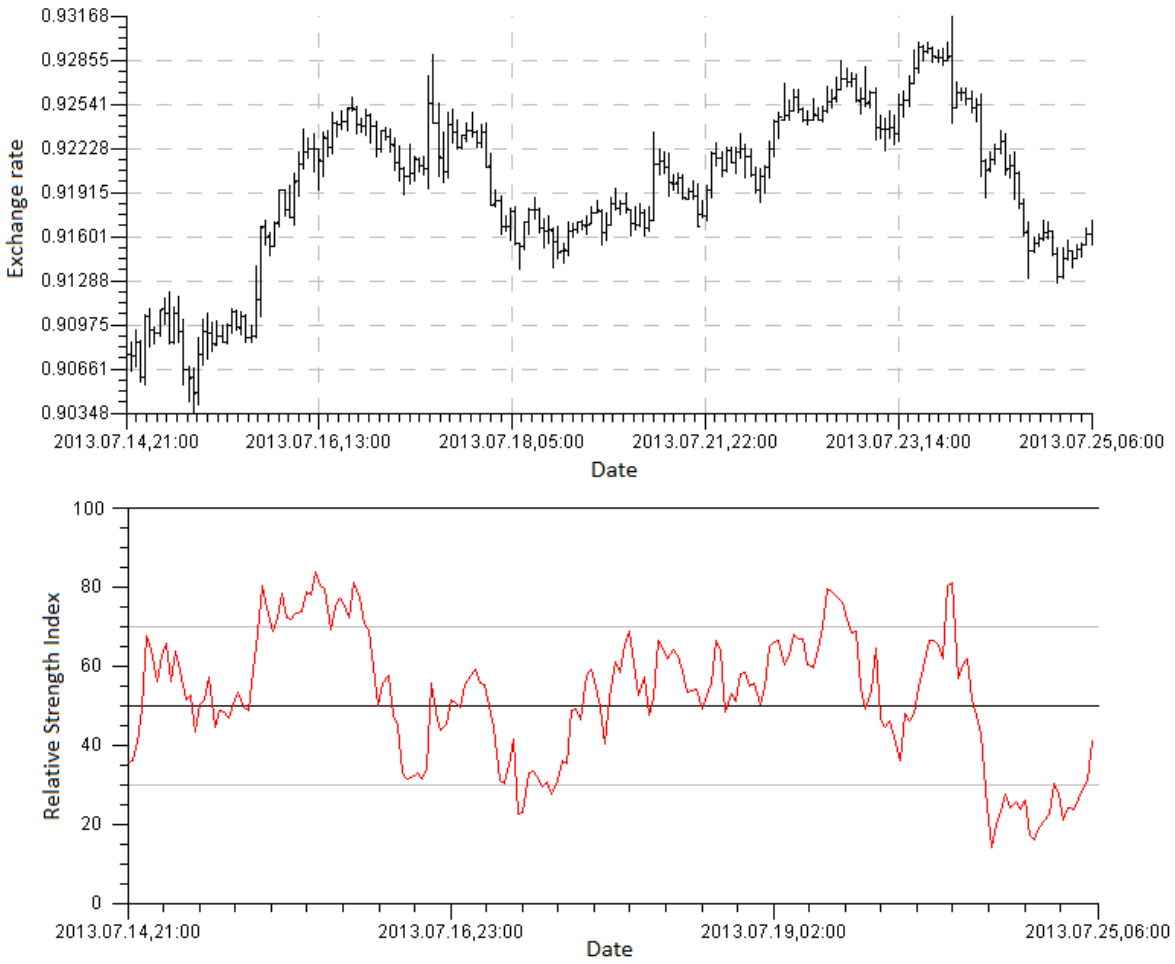


Figure 33: Candlestick chart of hourly AUDUSD data and an RSI indictor using the last 14 price changes.

3.3.2 Stochastic Oscillator (SO)

The stochastic oscillator (illustrated in Figure 34) is a momentum indicator promoted by Dr. George Lane in the 1950s. There is no contemporary academic reference, but Lane states that this was in 1954 (Lane, 1985).

The idea of the stochastic oscillator is that prices close near a recent high close price during bull markets, and close near a recent low close price during bear markets. From this, traders can get a rough idea of when the trend might be reversing and when the asset is overbought or oversold. The stochastic oscillator attempts to predict future price movement by considering where the current close price is compared to the highest and lowest close price of the last n close prices. It is found using the following formula (Bell, 2016):

$$\frac{P_{current} - P_{low}}{P_{high} - P_{low}} \times 100$$

where $P_{current}$ is the current close price, P_{low} is the lowest close price of the last n days, and P_{high} is the highest close price of the last n days.

When the SO value is above 80, this indicates that there is buying momentum and the market is overbought. If the SO value is below 20 then there is selling momentum and the market is oversold. The values 80 and 20 are typical thresholds and are normally the default settings on most trading platforms.

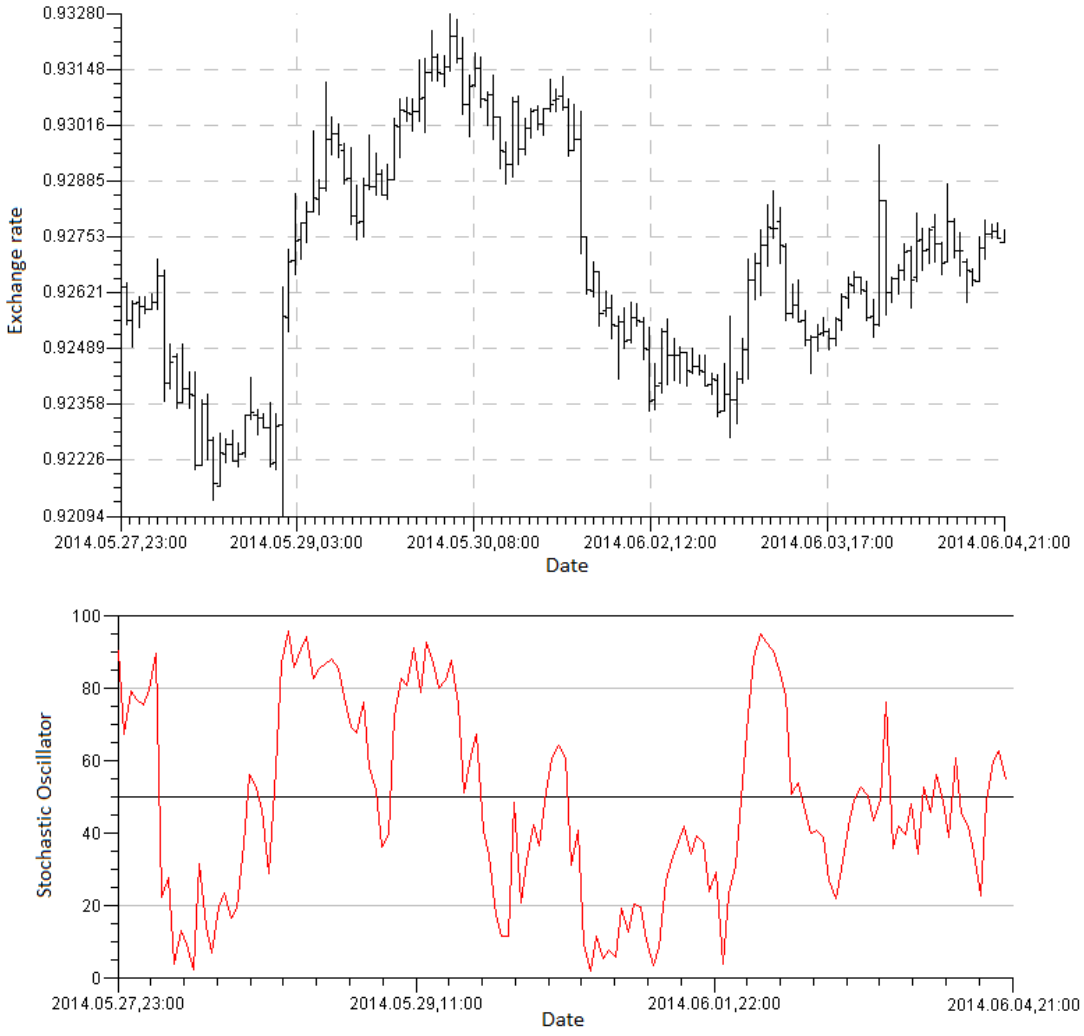


Figure 34: Candlestick chart of hourly AUDUSD data and an SO indictor using the last 14 price changes. Overbought and oversold levels are SO values of above 80 and below 20 respectively

3.3.3 Momentum Interpretations

The output of technical analysis algorithms that model momentum can be interpreted in various ways. Here are the interpretations implemented in this thesis for the stochastic oscillator and the relative strength index.

1. Buying momentum if the oscillator value is above a certain value. Selling momentum if the oscillator value is below a certain value.
2. Buying momentum if last oscillator value is below the current oscillator value. Selling momentum if last oscillator value is above the current oscillator value.
3. Buying momentum if the oscillator value is above a certain value for the last x data points. Selling momentum if the oscillator value is below a certain value for the last x data points.
4. Buying momentum if the oscillator value is above the last x oscillator values. Selling momentum if the oscillator value is below the last x oscillator values.
5. Buying momentum if the oscillator value is above the average of the last x oscillator values. Selling momentum if the oscillator value is below the average of the last x oscillator values.

Chapter 4 – Metrics for Evaluating Performance

Market participants are continually seeking ways to make money from the financial markets by finding profitable trading strategies. Market participants are able to translate their trading system into code. The benefit of this is that the system can trade autonomously, working without bias from human emotion and in a manner that enables *backtesting*. Before deploying a trading strategy on the live market, it is good practice for market participants to test their trading strategy on historical data, a procedure called backtesting. Backtesting can help estimate a trading strategy's potential on unseen market data if done correctly and this is usually done by interpreting the performance metrics of the trading strategy.

Trading strategies can also be optimised on *insample market data* by maximising performance metrics such as the Sharpe ratio, see 4.1. Then the trading strategies can be evaluated by inspecting the performance metrics on the market data continuing from the insample market data called *validation market data.*

Performance metrics are critical in the evaluation of trading strategies and traders as they quantify risk, profit and general performance. The performance metrics used in this thesis are outlined in this chapter. While other metrics are described in the literature, those chosen are the most common in trading software (such as e.g. *MetaTrader*), and the most likely to be recognised and trusted by traders.

Exceptional are the mean squared error (MSE), mean squared root error (RMSE), and mean absolute error (MAE). Although these are well-established measures which have been used by e.g. Khalifehloo *et al*, they do not furnish more decision-making information than Pearson's coefficient (below).

Note that other performance metrics are described in the literature. The list used in the thesis is not exhaustive as there are too many to implement. The most commonly given performance metrics in trading software such as MetaTrader were chosen.

4.1 Sharpe Ratio

The Sharpe ratio (Sharpe W. F., 1966; Sharpe W. F., 1994) is a risk-adjusted return measure which is used to compare investments against a risk free asset, and is defined by the following formula,

$$\frac{R - r}{\sigma_R}$$

where R is the expected return of an investment, r is the return on a risk free asset and hence equal to the current interest rate, and σ_R is the standard deviation of returns of the investment R.

If an investment has a positive Sharpe ratio, then the ratio indicates that the returns should outperform a risk free asset such as a US treasury bound. If the Sharpe ratio is negative then the risk free asset is the better investment.

The Sharpe ratio is similar to the inverse of the coefficient of variation formula. The formula, in the context of investment returns, indicates how much return is expected per unit risk,

$$\frac{\mu}{\sigma}$$

where μ is the mean and σ is the standard deviation.

To maximise the inverse coefficient of variation or the Sharpe ratio, an investment or trading strategy needs to produce high returns with low risk.

4.2 Sortino Ratio

Another popular performance metric is the Sortino ratio (Sortino, 1994). This ratio is similar to the Sharpe ratio but uses the standard deviation of only the negative returns instead of for both positive and negative returns:

$$\frac{R - r}{\sigma_{R_{neg}}}$$

where, R is the expected return of an investment, r is the return of a risk free asset, and $\sigma_{R_{neg}}$ is the standard deviation of negative returns of the investment R.

By using the standard deviation of negative returns, the ratio focuses on downside risk and not the risk of all returns.

4.3 Return Prediction Errors

Mean squared error (MSE), mean squared root error (RMSE), and mean absolute error (MAE) are traditional statistical methods for calculating the performance of a trading system. Though this thesis does not use these performance measures, they have been used in the creation of machine-learning based trading systems (Khalifehloo, Habibi, Mohammad, & Heydari, 2017). The formulas are:

$$MSE = \frac{1}{N}\sum_{t=1}^{N}\left(\hat{Y}_t - Y_t\right)^2$$

$$RMSE = \sqrt{\frac{1}{N}\sum_{t=1}^{N}\left(\hat{Y}_t - Y_t\right)^2}$$

$$MAE = \frac{1}{N}\sum_{t=1}^{N}\left|\hat{Y}_t - Y_t\right|$$

where N is the number of data points, Y_i is the predicted value from some model (such as the return values from a trading strategy) and $\hat{Y}_i$ is the observed value (such as historical close price differences).

To evaluate the performance of a trading system using the mean squared error formula, the difference in the predicted returns of the trading system and the asset's actual returns is calculated. The average of all the square differences produces an error value that indicates how well the trading system predicted the return. If the mean squared error is 0 then the trading system perfectly predicted the future returns of the asset. The further away the result is from 0, the greater the forecasting error. By comparing values of several trading strategies it is possible to pick the 'best' strategy with the least amount of error.

In forecasting systems such as an artificial neural networks, the mean squared error can be used to evaluate the neural network's performance during the optimisation of its weights and to quantitatively judge different forecasting systems (Chaudhuri & Ghosh, 2016).

4.4 Pearson's Correlation

Yümlü, Gürgen and Okay (2005) examines the correlation between a trading system's predicted returns and the observed returns of the asset. Pearson's correlation formula is as follows,

$$r_{xy} = \frac{\sum_{t=1}^{N}(x_t - \hat{x})(y_t - \hat{y})}{\sqrt{\sum_{t=1}^{N}(x_t - \hat{x})^2 \sum_{t=1}^{N}(y_t - \hat{y})^2}}$$

where N is the number of data points, x_i is the predicted return, $\hat{x}$ is the mean of the predicted returns, y_i is the observed return and $\hat{y}$ is the mean of the observed returns.

4.5 Theil's Inequality Coefficient

The Theil inequality coefficients decompose the mean square error into three components: bias U^M, unequal variation U^S and unequal covariation U^C (Watson & Teelucksingh, 2002). Each quantifies a different source of error. Of these, bias U^M measures systematic error which can typically be corrected by adjustment of parameters, and is therefore useful in evaluating and correcting the performance of a trading system (assuming that this has been correctly modelled).

4.6 Typical Performance Metrics in Real-World Trading

The "performance metrics" noted in Table 3, are used by traders to help determine the future performance of their trading strategies. These performance metrics are obtainable from backtesting reports such those obtained by *MetaTrader 4* (MetaQuotes Software), which is a software tool that enables buying and selling on financial markets and the ability to employ algorithmic trading strategies. Also there are features for optimising trading strategy parameters.

This tool is used widely in trading, making it a good starting point for understanding those performance metrics commonly used; these are shown in Table 3. Although traders use these metrics to judge whether or not a trading system is likely to perform well in the future, they are in all cases retrospective, indicating only how well the trader is performing at the moment.

Nevertheless, their currency means that any model is likely to be judged against them, and several will be used in this thesis.

Metric	Description
Gross profit	Total profit from winning trades
Gross loss	Total losses from losing trades
Total net profit	Total return from all trades
Profit factor	The ratio of gross profit to gross loss
Expected payoff	The average expected return per trade
Absolute drawdown	Largest loss from initial bankroll
Maximal drawdown	Largest bankroll loss
Relative drawdown	Largest bankroll loss expressed as a percentage of initial bankroll
Total trades	Total number of trades executed
Short positions	Total number of sell trades
Percentage of short positions won	Total number of profitable sell trades
Long positions	Total number of buy trades
Percentage of long positions won	Total number of profitable buy trades
Profitable trades	Total number of profitable trades
Percentage of trades that were profitable	Total number of profitable trades expressed as a percentage of total trades
Unprofitable trades	Total number of unprofitable trades
Percentage of trades that were unprofitable	Total number of unprofitable trades expressed as a percentage of total trades
Largest profit trade	Largest profit from a single trade
Largest loss trade	Largest loss from a single trade
Average profit trade	Average profit from a profitable trade
Average loss trade	Average loss from a unprofitable trade
Maximum consecutive wins	Longest number of profitable consecutive trades
Maximum consecutive losses	Longest number of unprofitable consecutive trades
Maximal consecutive profit	The largest profit obtained during consecutive trades
Maximal consecutive loss	The largest loss obtained during consecutive trades
Average winning streak	The average number of consecutive trades that were profitable

Average losing streak	The average number of consecutive trades that were unprofitable

Table 3: MetaTrader 4 performance metrics

4.7 Use of Performance Metrics in Machine Learning

In machine learning, trading systems such as artificial neural networks and support vector machines can also be optimised by maximising a performance metric such as the Sharpe ratio or Sortino ratio. Other authors' use of such performance metrics is summarised in Table 4.

Authors	**Performance Metric**
Nóbrega and Oliveira (2013)	Sharpe ratio, root mean squared error, return, volatility and the Theil inequality coefficient. The authors the metrics to compare the performance of support vector regression and their extreme learning machine forecasting models with varying linear regression models.
Evans, Pappas and Xhafa (2013)	Sharpe ratio, annualized return, trade success rate (prediction accuracy) and mean absolute error calculations. The authors report 72.5% prediction accuracy and a 23.3% return on foreign exchange intra-data market data by using neural networks and Genetic Algorithms.
Qiu, Song, & Akagi (2016)	Mean squared error The authors used a mean squared error

	function to evaluate the performance of their artificial neural network on the Nikkei 225 index. The mean squared error compared the returns of the Nikkei 225 index and the predicted return of their artificial neural network.
Zhang, Hu, Xie, Wang, Ngai, & Liu (2014)	Return, Sharpe ratio, Sortino ratio, annualised return and maximum drawdown. The authors used the performance metrics to compare different feature selection algorithms and different machine learning algorithms.

Table 4: Performance metrics in literature

4.8 Evaluations of Trading Success in this Thesis

There are longstanding difficulties with evaluating trading success. Stewart studied grain futures over a 9-year period and found that 75% of speculators lost money (Stewart, 1978). Hieronymus (1977) studied the closed trades for a single commission house and found that 65% of the firm's customers ended the year 1969 with a loss.

The author's study is not concerned with exploiting trader behaviours as much as identifying consistent trends and patterns of these. However, (Barbe & Odean, 2000) presents evidence that "overconfidence" leads to excessive trading, in which the better traders would be the more cautious ones. Evidence to this effect is being gathered, but remains essentially anecdotal.

In order for a trading strategy to make money reliably, it must perform well across a number of performance criteria, including return, risk, number of trades and trade success rate. A trading strategy which has been optimised only for profit without considering other performance metrics could, for example, trade too often, be extremely risky, and/or be overfitted. It is, however, difficult to find the best trade-off between performance metrics

that can reliably assess trading strategy and trader performance. Table 5 shows the performance metrics implemented and used in this thesis.

	Metric Name	Formula	Notes
1	Profit factor	$\frac{p}{l}$	p is profit, l is loss
2	Normalized profit factor	$\frac{p}{p+l+1}$	p is profit, l is loss
3	Maximal Drawdown	D	D the decline in the profit of a trading strategy between its maximum and minimum value, over a given period.
4	Return	R	R the total return
5	Average trade return	R_{avg}	R_{avg} the average return
6	Standard deviation of trade returns	R_{σ}	R_{σ} the standard deviation of returns
7	Number of trades	T_{total}	T_{total} the total number of trades
8	Number of unsuccessful trades	T_{lost}	T_{lost} the total number of unprofitable trades
9	Number of successful trades	T_{won}	T_{won} the total number of profitable trades
10	Trade success rate	$\frac{T_{won}}{T_{lost}}$	T_{won} is the total number of profitable trades and T_{lost} the total number of unprofitable trades

11	Average winning streak	TS_{avgWin}	TS_{avgWin} The average number of consecutive profitable trades
12	Average losing streak	$TS_{avgLost}$	$TS_{avgLost}$ the average number of consecutive unprofitable trades
13	Normalized average win/lose streak:	$\frac{TS_{avgWin}}{TS_{avgWin} + TS_{avgLost} + 1}$	TS_{avgWin} The average number of consecutive profitable trades. $TS_{avgLost}$ the average number of consecutive unprofitable
14	Longest winning streak	$TS_{longestWin}$	$TS_{longestWin}$ the longest number of consecutive profitable trades
15	Longest losing streak	$TS_{longestLost}$	$TS_{longestLost}$ the longest number of consecutive unprofitable trades
16	Normalized longest win/lose streak	$\frac{TS_{longestWin}}{TS_{longestWin} + TS_{longestLost} + 1}$	$TS_{longestWin}$ the longest number of consecutive profitable trades. $TS_{longestLost}$ the longest number of consecutive unprofitable trades
17	Standard deviation of winning streaks	$TS_{\sigma Wins}$	$TS_{\sigma Wins}$ the standard deviation of winning streak lengths
18	Standard deviation of losing streaks	$TS_{\sigma Losses}$	$TS_{\sigma Losses}$ the standard deviation of losing streak lengths
19	Largest trade	$T_{largestProfit}$	$T_{largestProfit}$ the largest

	profit		profitable trade
20	Largest trade loss	$T_{largestLoss}$	$T_{largestLoss}$ the largest unprofitable trade
21	Average profitable trade	$T_{avgProfit}$	$T_{avgProfit}$ the average profitable trade
22	Average unprofitable trade	$T_{avgLoss}$	$T_{avgLoss}$ the average unprofitable trade
23	Standard deviation of profitable trades	$T_{\sigma Profits}$	$T_{\sigma Profits}$ the standard deviation of profitable trades
24	Standard deviation of unprofitable trades	$T_{\sigma Losses}$	$T_{\sigma Losses}$ the standard deviation of unprofitable trades
25	Longest streak ratio	$\frac{TS_{longestWin}}{TS_{longestLoss}}$	$TS_{longestWin}$ the longest winning streak. $TS_{longestLoss}$ the longest losing streak
26	Average streak ratio	$\frac{TS_{avgWin}}{TS_{avgLoss}}$	TS_{avgWin} the average winning streak length. $TS_{avgLoss}$ the average losing streak length
27	Standard deviation streak ratio	$\frac{TS_{\sigma Win}}{TS_{\sigma Loss}}$	$TS_{\sigma Win}$ the standard deviation of winning streak lengths. $TS_{\sigma Loss}$ the standard deviation of losing streak lengths
28	Largest trade ratio	$\frac{T_{largestProfit}}{T_{largestLoss}}$	$T_{largestProfit}$ the largest profitable trade. $T_{largestLoss}$ the largest unprofitable trade

29	Average trade ratio	$\frac{T_{avgWin}}{T_{avgLoss}}$	T_{avgWin} the average profitable trade. $T_{avgLoss}$ the average unprofitable trade
30	Standard deviation trade ratio	$\frac{T_{\sigma Win}}{T_{\sigma Loss}}$	$T_{\sigma Win}$ the standard deviation of profitable trades. $T_{\sigma Loss}$ the standard deviation unprofitable trades
31	Sharpe ratio	$\frac{T_{avg}}{T_{\sigma}}$	T_{avg} the average trade return. T_{σ} the standard deviation of trade returns
32	Sortino ratio	$\frac{T_{avg}}{T_{\sigma Losses}}$	T_{avg} the average trade return. $T_{\sigma Losses}$ the standard deviation of unprofitable trades
33	Winning Sortino ratio	$\frac{T_{avg}}{T_{\sigma Profits}}$	T_{avg} the average trade return. $T_{\sigma Profits}$ the standard deviation of profitable trades
34	Expected payoff	$\frac{R}{T_{total}}$	R is the return. T_{total} is the total number of trades.
35	Maximal consecutive profit	$TS_{biggestProfit}$	$TS_{biggestProfit}$ is the biggest profit made from consecutive trades
36	Maximal consecutive loss	$TS_{biggestLoss}$	

Figure 40 shows the relationship between the number of trades of each technical analysis interpretation on historical data against those on future market data. There is a strong positive correlation between the number of trades on historical and future market data of 0.997. The number of trades that a technical analysis interpretation places on historical data is therefore very likely to be similar to the number of trades placed on future market data over the same number of data points.

In Figure 40, there is a cluster of technical analysis interpretations that execute more than 2,250 trades on each of the historical and future datasets. This cluster of interpretations makes up 6% of the total set of interpretations. There seem to be clusters of technical analysis interpretations with the same number of trades on historical and future market data. Figure 41 shows the relationship between the trade success rate of each technical analysis interpretation and the number of trades it executes using historical data. This figure shows there are clusters of technical analysis interpretations with the same trade success rate and number of trades. This perhaps indicates again that different technical analysis interpretations can respond similarly to market phenomena.

Transaction costs are incurred by buying or selling assets, and so technical analysis interpretations that execute too many trades face large transaction costs which reduce overall profit. In total, 81% of the technical analysis interpretations make 200 or fewer trades on each hourly dataset, which spans 5,172 hours. On average, 20% of the technical analysis interpretations execute a trade every 26 hours to 52 hours and 61% execute trades longer than 52 hours apart. As the correlation between the number of trades of a technical analysis interpretation on historical and future market data is strong and positive, then it is possible to reduce transaction costs by selecting technical analysis interpretations that trade less often on historical data. Choosing technical analysis interpretations that have an appropriate number of trades on historical data is vital so that confidence can be placed on their analysis.

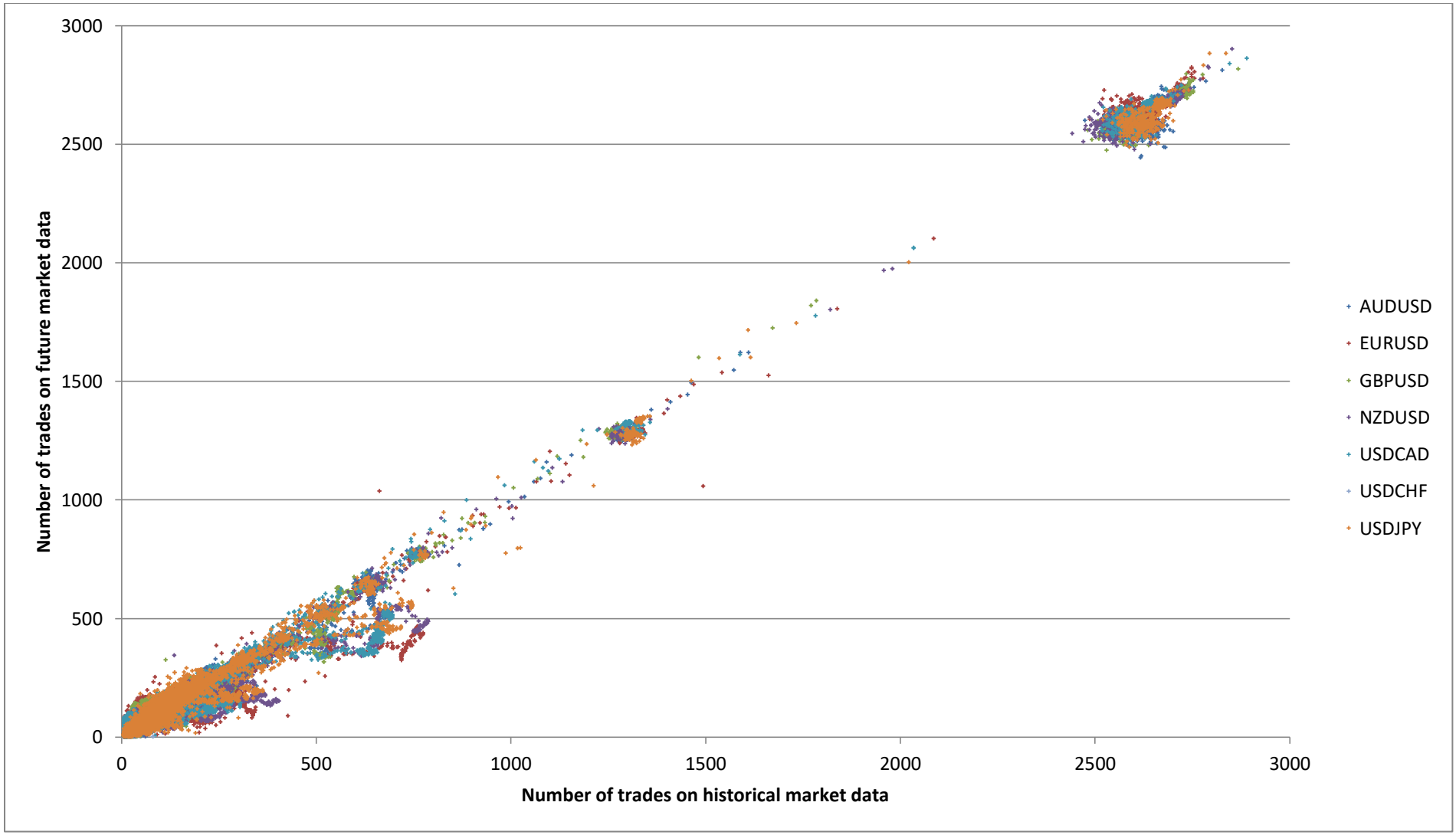


Figure 40: Relationship between each technical analysis interpretation's number of trades on historical and future market data

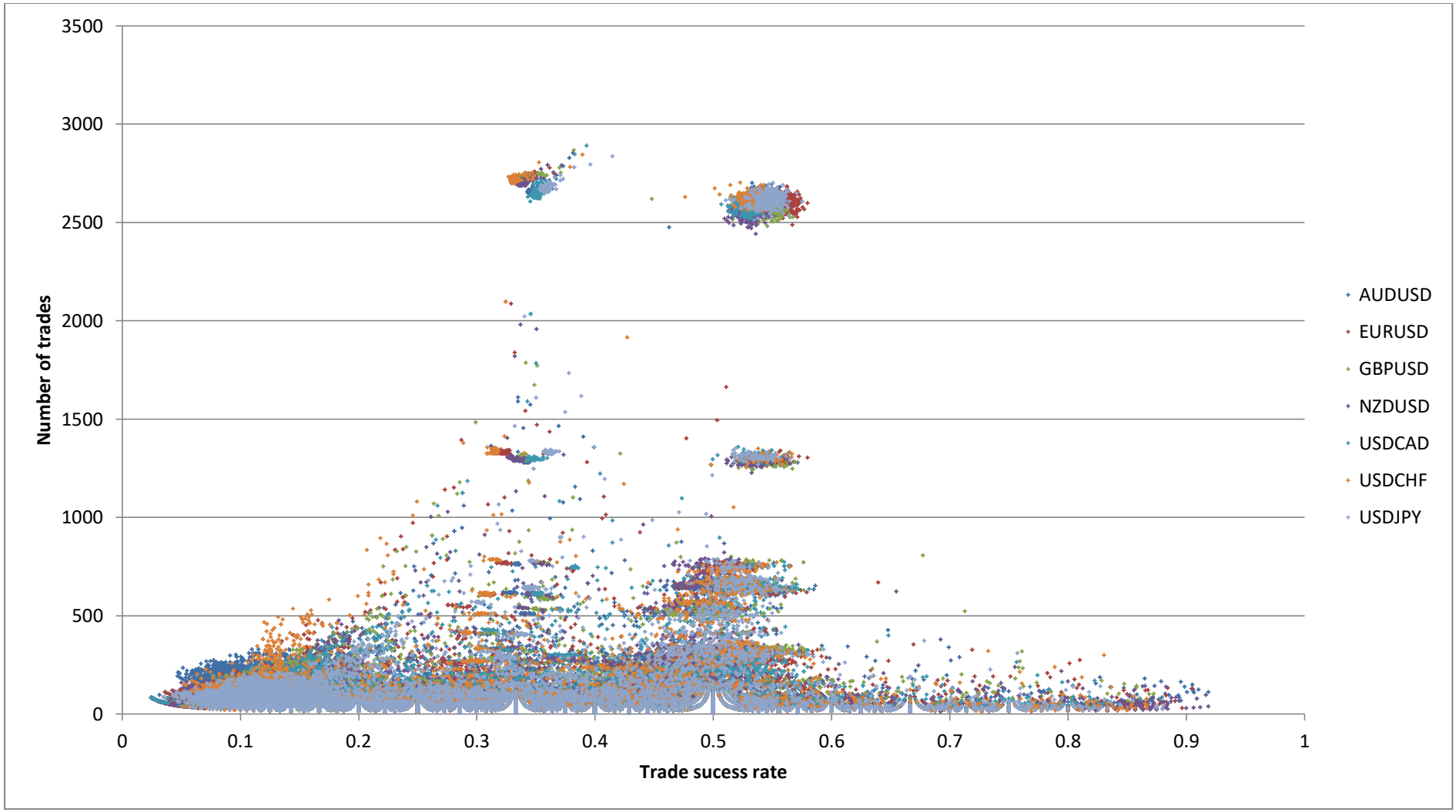


Figure 41: The relationship between each technical analysis interpretation's trade success rate and the number of trades on historical data

5.2 Random Generation of Trading Strategies

The previous section analysed the performance of technical analysis interpretations. This section applies the same analysis to the trading strategy model, which combines technical analysis interpretations. First, the structure of this combination is explained.

5.2.1 Trading Strategy Architecture

Traders commonly combine technical analysis interpretations to produce profitable trading strategies, reaching decisions to buy and sell by taking into account the output of each of the interpretations. To mimic real traders, trading strategies in this thesis are represented by a decision tree, see Figure 42. The terminating nodes (in red) encapsulate technical analysis interpretations, and direct their predictions of buy, sell or do nothing to a decision node (in blue). The decision node performs a majority vote decision on the technical analysis interpretations' outputs. If there is no majority decision, then the output defaults to do nothing. This representation represents the likely decision making process of a trader using a set of technical analysis interpretations

The trading strategies model is therefore a simplification of a trader. The size of a trading strategy is denoted by the number of technical analysis interpretations present.

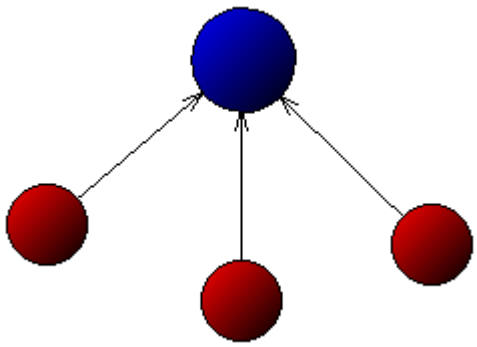


Figure 42: Representation of a trading strategy

5.2.2 Forecasting Potential of Trading Strategies

Figures 43 and 44 show the distributions of return for different sizes of trading strategies. These were derived from the AUDUSD historical and future market dataset. For each trading strategy size, the returns are calculated from historical AUDUSD market data by the random

generation of 10,000 trading strategies which have at least 5 trades, as before. The historical distribution of returns implies that the more technical analysis interpretations are present in the model, the greater the chance for profit. The distribution of returns derived from the future market dataset, however, shows that there is no difference in profit between the different sizes of trading strategies. The return distributions from the other foreign exchange markets demonstrated that more technical analysis interpretations could even lead to a worse return.

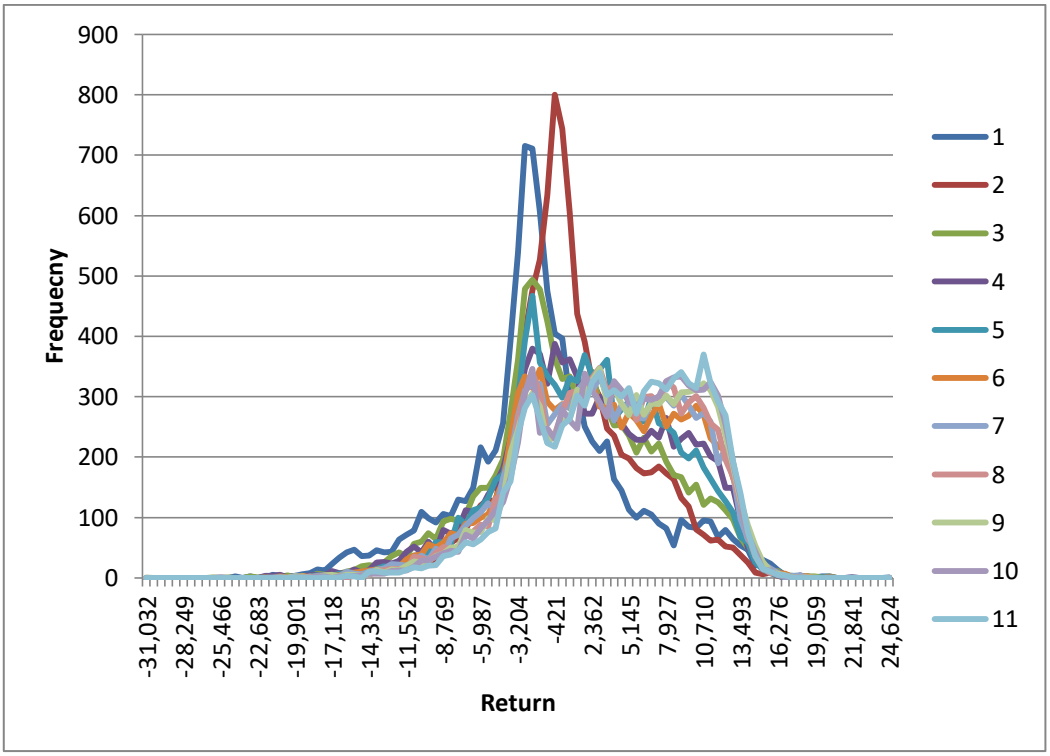


Figure 43: Returns of 10,000 trading strategies with varying sizes (from 1 to 11) on historical AUDUSD market data

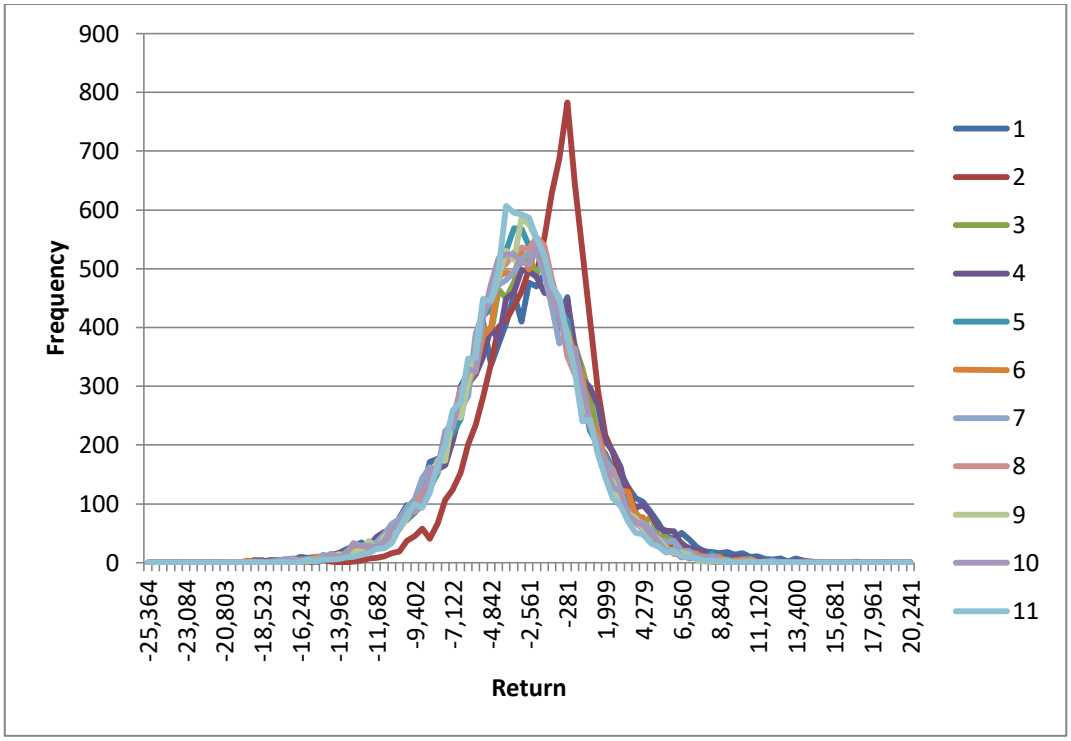


Figure 44: Returns of 10,000 trading strategies with varying sizes (from 1 to 11) on future AUDUSD market data

Figure 45 shows the average return generated by each trading strategy size for each historical and future market dataset. The figure shows the average profit or loss is magnified as the number of technical analysis interpretations in the trading strategy increases. Additionally, the average return of trading strategies using the historical market datasets does not reflect the average return on future market datasets.

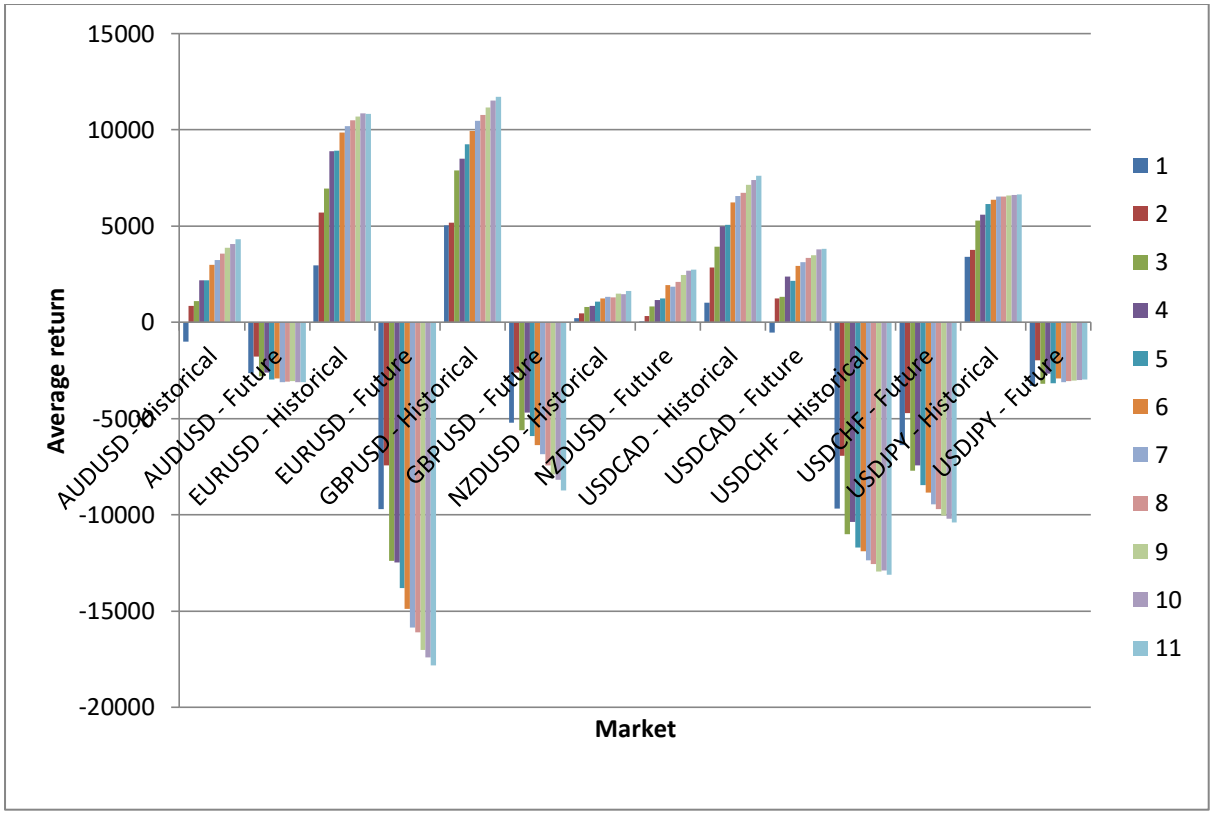


Figure 45: Average return of each trading strategy size for each market segment

Tables 9 and 10 show the average correlation between performance metrics of trading strategies on historical and future market datasets using trading strategies of size 2 and 11 respectively; 'heat' colours emphasize the level of correlation , in order from dark green (highest) through light green and yellow to orange (lowest). The average correlation values produced by both trading strategy sizes are similar; the returns of trading strategies of size 2 have a mean of 0.0296 and a standard deviation of 0.3159 and the returns of trading strategies of size 11 have a mean of 0.0243 and a standard deviation of 0.3024. There are differences between the average correlations of trading strategies of size 2 and the average correlations of single technical analysis interpretations in Table 8 outlined in the Section 5.1.

By comparing single technical analysis interpretations with trading strategies of size 2, new correlations emerge. The standard deviation of returns metric on historical data has a moderate positive correlation with the standard deviation of returns, a weak inverse correlation with the Sharpe ratio metric, a weak positive correlation with the Sortino ratio metric and a moderate inverse correlation with the trade success rate metric on future market data. The trade success rate metric using historical data has a moderate inverse correlation with the standard deviation of return metric, a weak correlation with the Sharpe

ratio metric and a weak inverse correlation with the Sortino ratio metric on future market data. There is a weak correlation of 0.29 between the trade success rate metric from historical data and the return metric from future data using single technical analysis interpretations, and for trading strategies with 2 technical analysis interpretations there is only a correlation of 0.19.

	Return	Standard deviation of returns	Sharpe ratio	Sortino ratio	Trade success rate
Return	-0.004859845	0.150953643	-0.009691731	0.005766127	-0.016167893
Standard deviation of returns	-0.149940529	0.640079523	-0.321770071	0.292021906	-0.591804065
Sharpe ratio	0.001343334	0.165834961	0.029404454	-0.035421465	-0.039597172
Sortino ratio	0.08089998	-0.217240701	0.140891941	-0.122043822	0.206767018
Trade success rate	0.190081127	-0.522794418	0.323085371	-0.318795214	0.863110102

Table 9: Average correlation between metrics of trading strategies of size 2 from historical and future market data

	Return	Standard deviation of returns	Sharpe ratio	Sortino ratio	Trade success rate
Return	-0.007843202	0.064975802	0.007804762	-0.015849052	0.042753183
Standard deviation of returns	-0.141317054	0.575655138	-0.326920418	0.317257195	-0.545087243
Sharpe ratio	-0.02209507	0.121535358	0.011818742	-0.025494459	-0.022822122
Sortino ratio	0.087568906	-0.179635965	0.141928404	-0.121615343	0.169035726
Trade success rate	0.16840483	-0.517964453	0.317372717	-0.338790557	0.845604215

Table 10: Average correlation between metrics of trading strategies of size 11 from historical and future market data

5.3 Conclusions

More correlations emerged when using a trading strategy consisting of multiple technical analysis interpretations when using one technical analysis interpretation.

Maximising and minimising historical performance metrics may lead to better performing trading strategies, given the correlation values between the historical and future market datasets. In fact, traders attempt to find 'good' trading strategies by minimising and

maximising performance metrics using historical market data. These correlations suggest some historical performance metrics may have influence on the future performance of trading strategies. However, some correlations have been shown to be weak, suggesting that there may be a benefit in combining several performance metrics into one ensemble method that demonstrates a stronger correlation (i.e. predicts a given metric more reliably). The performance metrics identified in Chapter 4 will be used in subsequent chapters, and the idea of combining performance metrics is explored in Chapter 8.

Chapter 6 – Generating Trading Strategies using Genetic Algorithms

In order ultimately to produce and test a classification system for identifying 'bad' traders, it is necessary first to have some method for producing a pool of trading strategies that mimic traders. Traders deploy trading strategies either by manually executing trades or deploying automated trading systems. A trader's performance is directly linked to the performance of the trading strategies that they deploy. Essentially, in the context of this thesis, traders are trading strategies.

This chapter explores the generation of trading strategies that traders are likely to deploy, i.e. ones demonstrating effective performance on historical data.

6.1 Lipinski's Research

While much can be found on the creation and optimisation of trading strategies in the literature, very little research is evident in the literature specifically on the combination of technical analysis algorithms into trading strategies without optimising the technical analysis algorithm's parameters. The work of one author, Lipinski et al (2010; 2004; 2010), stands out in this regard. As the concept of the trading strategy is to be used in this thesis as proxy for a trader, the work of that author in Genetic Algorithms offers a good starting point for the generation of a pool of effective trading strategies in this thesis. The results of Lipinski are therefore reviewed here.

Lipinski represents a trading strategy by a binary chromosome, in which each gene indexes a technical analysis algorithm (as mentioned in Section 2.7.1). If a gene has the value 1, then the technical analysis algorithm indexed by that gene is activated; if the value is 0 then the technical analysis algorithm is deactivated. The technical analysis algorithms that are activated produce buy and sell signals, and the trading strategy containing these technical analysis algorithms takes the majority vote of their buy or sell signals to produce the final buy or sell decision.

Their trading strategy representation has no direct control over how many trading rules are activated in each trading strategy; the evolutionary process dictates how many are activated in each chromosome. The implemented representation of Lipinski et al. also limits the total number of genes in a chromosome so that only a predefined set of trading rules are considered. Additionally, depending on the crossover algorithm used in the Genetic

Algorithm, for example, the single point crossover, the crossover operation is likely to break good combinations depending on their positions within the chromosome. In reality traders are more likely to deploy trading strategies that contain only a few technical analysis algorithms to reduce the chance of overfitting on historical market data. With the approach of Lipinski et al., too many technical analysis algorithms can contribute to the final decision of the trading strategy, there is therefore an increased chance of overfitting the trading strategy by perfectly matching the trades of trading rules with the highs and lows of the historical training dataset.

Discussion will now turn in detail on how the system presented here overcomes the above problems by using a different representation and crossover operation with the focus being to create the type of trading strategy that traders are likely to create.

6.2 Genetic Algorithm Configuration

Genetic Algorithms, as outlined in Chapter 2, find local optima solutions to problems that have a vast search space and which cannot be completely enumerated. This section attempts to apply Genetic Algorithms to create profitable trading strategies by maximising the return performance metric. The goal of this experiment is to determine whether trading strategies of the type employed by human traders can be evolved to be as effective. Many parameter settings for the Genetic Algorithm are explored and each parameter set is ran 100 times to report on the parameter set's average performance.

The results in this chapter are derived from hourly AUDUSD foreign exchange data from 19th of May 2014 to the 23rd of January 2016. The hourly AUDUSD market dataset contains 10,344 data points.

The foreign exchange market is split into 3 market datasets as outlined in Table 11. During the evolutionary process of the Genetic Algorithm, the insample dataset is used by the trading strategies to calculate each trading strategy's fitness value. The validation dataset is a segment of market data that is traded by the trading strategies so that an optimal iteration for selecting the best trading strategy can be determined. The validation dataset also helps to confirm that the trading strategies in the population have shown to have predictive value outside the insample dataset. The outsample dataset is used by the trading strategies to test the performance of the trading strategies on future market data.

Market data	Date from	Data to	Number of data points
Insample	19th of May 2014	5th of December 2014	3,448
Validation	5th of December 2014	1st of July 2015	3,448
Outsample	1st of July 2015	23rd of January 2016	3,448

Table 11: Information about the datasets obtained from each foreign exchange market dataset

The main concern is that trading strategies may overfit the insample dataset, which occurs when the trading strategies memorise the insample dataset. The purpose of the validation dataset is to detect an optimal Genetic Algorithm iteration for choosing a trading strategy before the Genetic Algorithm memorises the insample data. Ideally, both insample and validation fitness values obtained from a fitness function (described in 6.2.3) should increase with iteration. Beyond the optimal iteration, the insample fitness values should then continue to increase or converge, whilst the same trading strategies on validation data converge or decrease in value. The fitness values obtained by trading the validation market data is not seen by the Genetic Algorithm and this decrease or convergence in fitness on validation data indicates overfitting where the Genetic Algorithm starts to memorise the insample dataset.

The following sections will discuss in detail the Genetic Algorithm implementation for the experiments in this chapter.

6.2.1 Representation

The representation of a trading strategy used in the Genetic Algorithm experiments was outlined in Section 5.2.1. Trading strategies consist of one or more technical analysis interpretations that produce buy and sell signals which are forwarded to a majority vote decision. Each technical analysis interpretation is a gene. The goal of this thesis is to create an early warning detection system for bad traders, and so the idea here is to generate trading strategies that replicate the decision making process a trader is likely to follow, given a set of technical analysis algorithms.

This topological representation fixes the number of technical analysis interpretations in each trading strategy to closer mimic traders and thus avoids the trading strategy becoming

overly complicated which can lead to overfitting. Traders are more likely to use only a few technical analysis interpretations per trading strategy.

6.2.2 Technical Analysis Interpretations

Technical analysis interpretations with random parameter settings were investigated in Section 5.2. Correlations found in that section suggest that parameter settings of technical analysis interpretations could be optimised based on historical metrics such as the Trade Success Rate metric.

In the following experiments, a pool of 10,000 technical analysis interpretations is created to form the initial pool of trading strategies, and also to be used as a source of new technical analysis interpretations during the mutation stage of the Genetic Algorithm. In total there are 33 different interpretations of technical analysis models implemented which are identified in Section Chapter 3. The pool of 10,000 technical analysis interpretations contains equal numbers of each interpretation implemented with the exception of an extra technical analysis interpretation randomly generated in the collection. Each technical analysis interpretation is created by random selection of parameter values, and only those that place at least five trades are included in the pool.

The trading strategy architecture restricts the number of technical analysis interpretations considered in a strategy, while the pool of 10,000 such interpretations ensures a healthy source of 'trading rules'.

6.2.3 Fitness Function

The fitness function is used to evaluate the performance of a trading strategy during the evolutionary process, and is also used during the selection process to evaluate the relative performance of trading strategies. Section 5.2 explored the correlations between a trading strategy's metrics on historical and future market data. In the following experiments, the Return performance metric will be used as the fitness function to maximise during the evolutionary process of the Genetic Algorithm. In the next chapter, which further investigates the creation of trading strategies using the Genetic Algorithm, explores using the Sharpe ratio, Sortino ratio and Trade Success Rate performance metrics as fitness functions in the Genetic Algorithm.

6.2.4 Initialisation

During the initialisation process, a population of trading strategies is created by combining random technical analysis interpretations from a pool of 10,000 randomly created technical analysis interpretations. Duplicate technical analysis interpretations can be found in a trading strategy but this is unlikely. Each trading strategy contains the same number of technical analysis interpretations and is given for each experiment.

6.2.5 Selection, Crossover and Mutation

During selection, fitness function values are calculated for each trading strategy. The five point universal sampling technique and tournament selection described in Section 2.7.1.3 and 2.7.1.4 respectively is used to select trading strategies for the crossover phase of the Genetic Algorithm. Trading strategies are selected until the number of trading strategies equals the fixed population size.

The topological representation of a trading strategy implemented in this thesis is intuitive and the crossover operation swaps a random technical analysis interpretation from each trading strategy. This enables already good combinations of technical analysis interpretations to stay together. An illustration of this is shown in Figures 46 and 47.

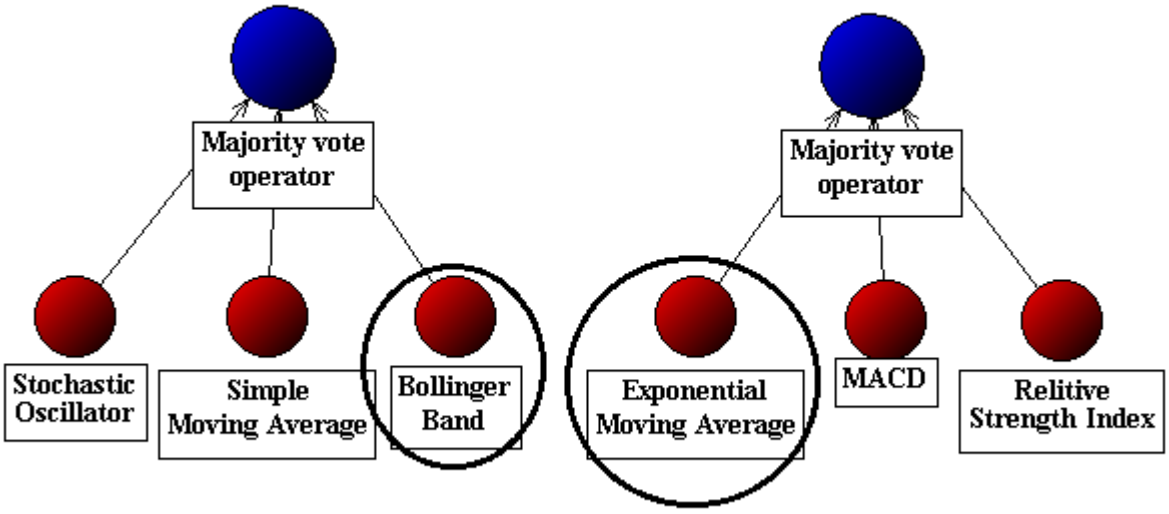


Figure 46: Two trading strategies each with a technical analysis interpretation selected to swap

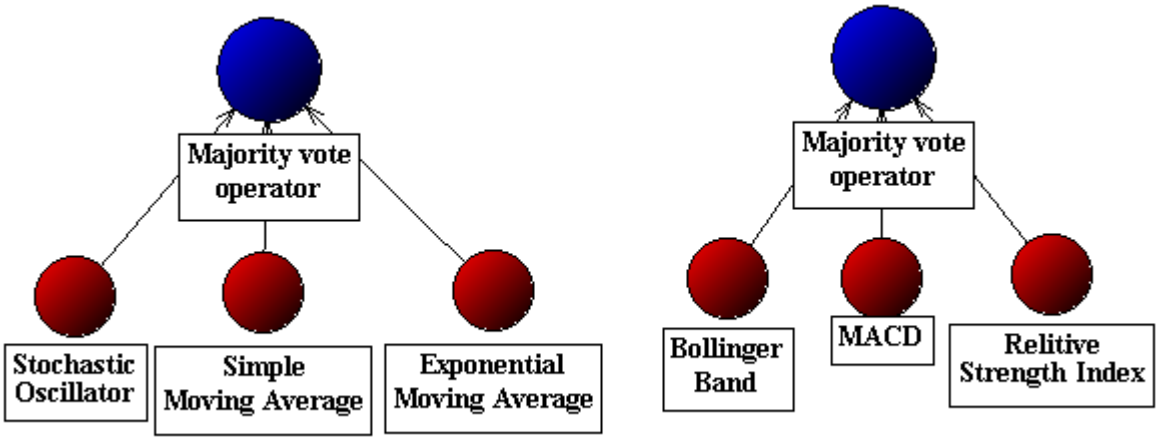


Figure 47: New trading strategies created by swapping technical analysis interpretations

Mutation introduces technical analysis interpretations which were not present in the initial population of trading strategies and helps avoid early population convergence. A trading strategy mutates given some probability, and if the trading strategy is to mutate then a random technical analysis interpretation is replaced by another technical analysis interpretation in the predefined pool of 10,000 random technical analysis interpretations. Duplicate technical analysis interpretations in a trading strategy may arise from mutation but it is unlikely.

6.2.6 Termination Condition

The population of trading strategies should generally increase in fitness at each iteration of the Genetic Algorithm. The Genetic Algorithm in the following experiments terminate after a predefined number of iterations, and then an optimal trading strategy can be chosen with low bias and low variance using the validation dataset as described in Section 2.3.

6.2.7 Genetic Algorithm Pseudocode

```
1.  SET 'taPool' TO a random set of 10,000 technical analysis interpretations
2.  DECLARE 'population'
3.
4.  FOR 'i' FROM 1 TO 300 DO
5.      DECLARE ‘tradingStrategy’
6.      ADD 3 random technical analysis interpretations from ‘taPool’ TO ‘tradingStrategy’
7.      ADD ‘tradingStrategy’ TO ‘population
8.  END DO
9.
10. FOR ‘gaIteration’ FROM 1 TO 100 DO
11.     DECLARE 'nextGeneration'
12.
13.     Calculate fitness values for each trading strategy in 'population'
14.     Normalise fitness values between 0 and 1
15.
16.     FOR 'numberOfTsAdded' FROM 1 TO 300 DO
17.         SET ‘randValue’ TO a random number from 0 up to but not including 1
18.         SET 'fitnessIndexValues' TO 5 values uniformly distributed from the value ‘randValue’ between 0 and 1
19.
20.         ADD trading strategies corresponding to the values in 'fitnessIndexValues' TO 'nextGeneration'
21.     END FOR
22.
23.     SET 'population' to an empty array
24.
25.     FOR 'w' FROM 1 TO 300 BY 2 DO
26.         SET 'rand' TO a random number from 0 up to but not including 1
27.
28.         SET 'strategy1' TO a copy of the trading strategy from 'nextGeneration' at index 'w'
29.         SET 'strategy2' TO a copy of the trading strategy from 'nextGeneration' at index ('w'+1)
```

```
30.
31.             IF 'rand' < 0.8 THEN
32.                     swap a random technical analysis interpretation in 'strategy1' with a random  technical analysis interpretation in 'strategy2'
33.             END IF
34.
35.             ADD 'strategy1' TO 'population'
36.             ADD 'strategy2' TO 'population'
37.     END FOR
38.
39.     FOR 's' FROM 1 TO 300 DO
40.             SET 'rand' TO a random number from 0 up to but not including 1
41.
42.             IF 'rand' < 0.01 THEN
43.                     SET 'tradingStrategy' TO the trading strategy from 'population' at index 's'
44.
45.                     replace technical analysis interpretation in 'tradingStrategy' with a technical analysis interpretation in 'taPool'
46.             END IF
47.     END FOR
48. END FOR
```

6.3 Results

This section investigates whether it is possible to use the Genetic Algorithm to create trading strategies that show predictive value on future data. Each experiment adjusts the Genetic Algorithm's configuration slightly to search for good Genetic Algorithm configurations.

Sources of randomness include:

- The pool of randomly created technical analysis interpretations;
- The population of randomly created trading strategies;
- Random selection process with a bias toward trading strategies with higher fitness;
- Random trading strategy crossovers;
- Random mutations.

The Genetic Algorithm inherently has random elements and produces different results in each run of the algorithm. To facilitate analysis, the Genetic Algorithm is run 100 times in order to generate average performance results.

6.3.1 Exploring the Parameter Settings of the Genetic Algorithm

Figure 48 shows the average performance of 100 Genetic Algorithm runs of the fittest trading strategy at each iteration of the Genetic Algorithm during using the insample dataset. The Genetic Algorithm uses the five point universal sampling selection technique and shifts the negative return values into the positive number domain for normalisation. The 12 different parameter setting configurations of the Genetic Algorithm which are explored are shown in Table 12 (For example, the dark blue line corresponds to a Genetic Algorithm configuration that uses a return fitness function, has a population size of 100, a crossover probability of 0.8 and a mutation rate of 0.1). The results show that the average returns of the fittest trading strategies generally reduce at each iteration of the Genetic Algorithm until they begin to oscillate below zero. The average maximum return of 100 runs is expected to increase at each iteration of the algorithm before the population converges or decreases in fitness. The iteration at which the average return converges or decreases in value is the optimal iteration before overfitting occurs.

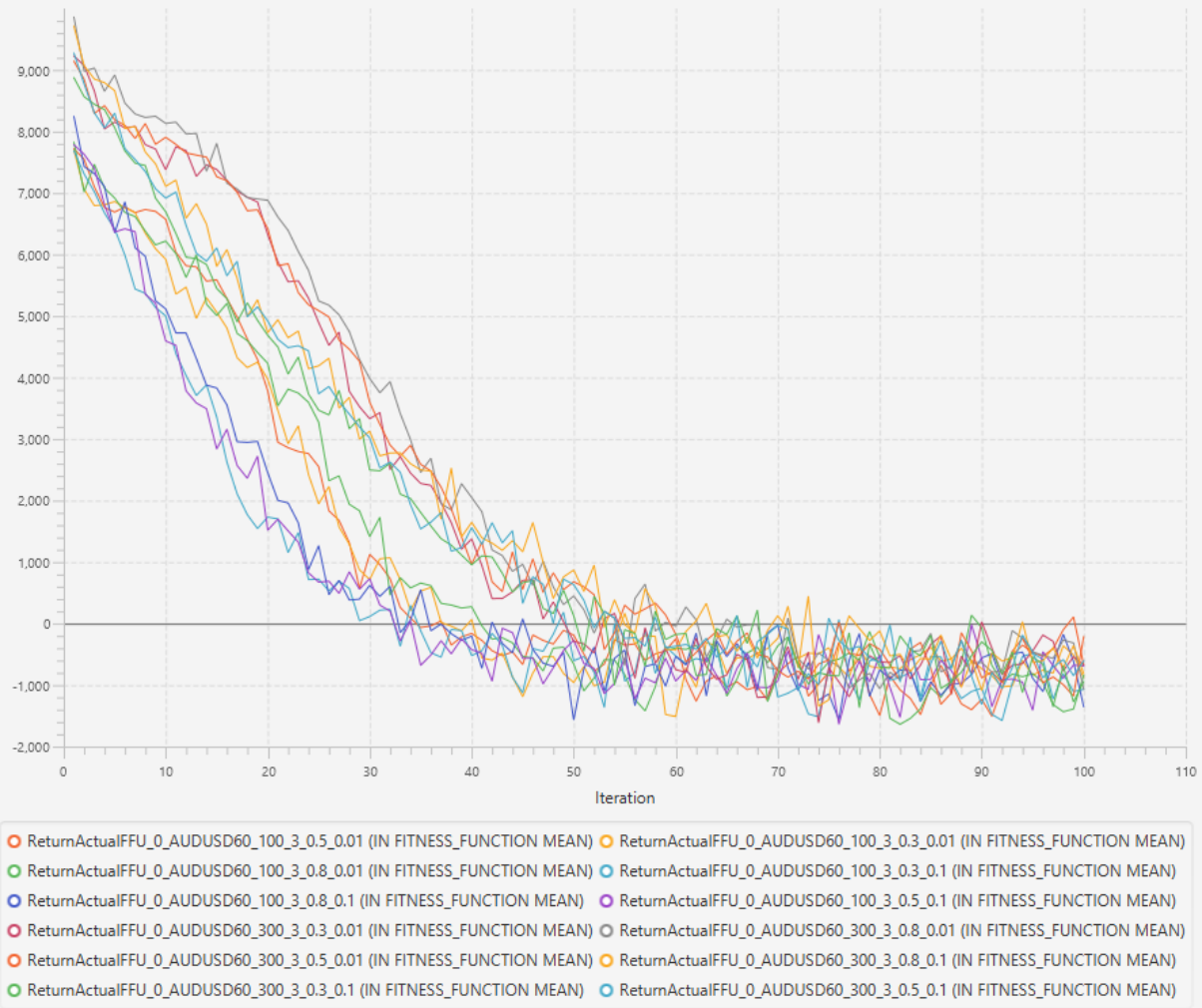


Figure 48: The average maximum return value at each iteration of the Genetic Algorithm using insample data

Stage	Parameter	Parameters explored
All	Size of technical analysis interpretation pool	10,000
All	Population size	100 and 300
All	Trading strategy size	3
Selection	Fitness function	Return
Crossover	Crossover probability	0.3, 0.5 and 0.8
Mutation	Mutation rate	0.01 and 0.1

Table 12: Genetic Algorithm parameter settings

Figures 49 and 50 show the average performance of 100 Genetic Algorithm runs of the fittest trading strategies from Figure 48 on validation and outsample data respectively. The fall and then levelling of performance of trading strategies using insample data is similar to the performance of the trading strategies on validation and outsample data. The average return drops during the first 30 – 50 iterations before oscillating below zero.

The validation dataset is used to spot the optimal iteration before the population begins to overfit. Though this is the purpose of the validation dataset, it can also be used as a form of out of sample data if the validation dataset cannot be used to spot optimal iterations. To pick profitable trading strategies on future data, these results suggest that the trading strategy with the highest return value on the insample dataset during the initial iterations of the Genetic Algorithm has the best chance of being the most profitable. However, the average return of the fittest trading strategies during the initial iterations of the Genetic Algorithm is high on validation data and roughly zero on outsample data.

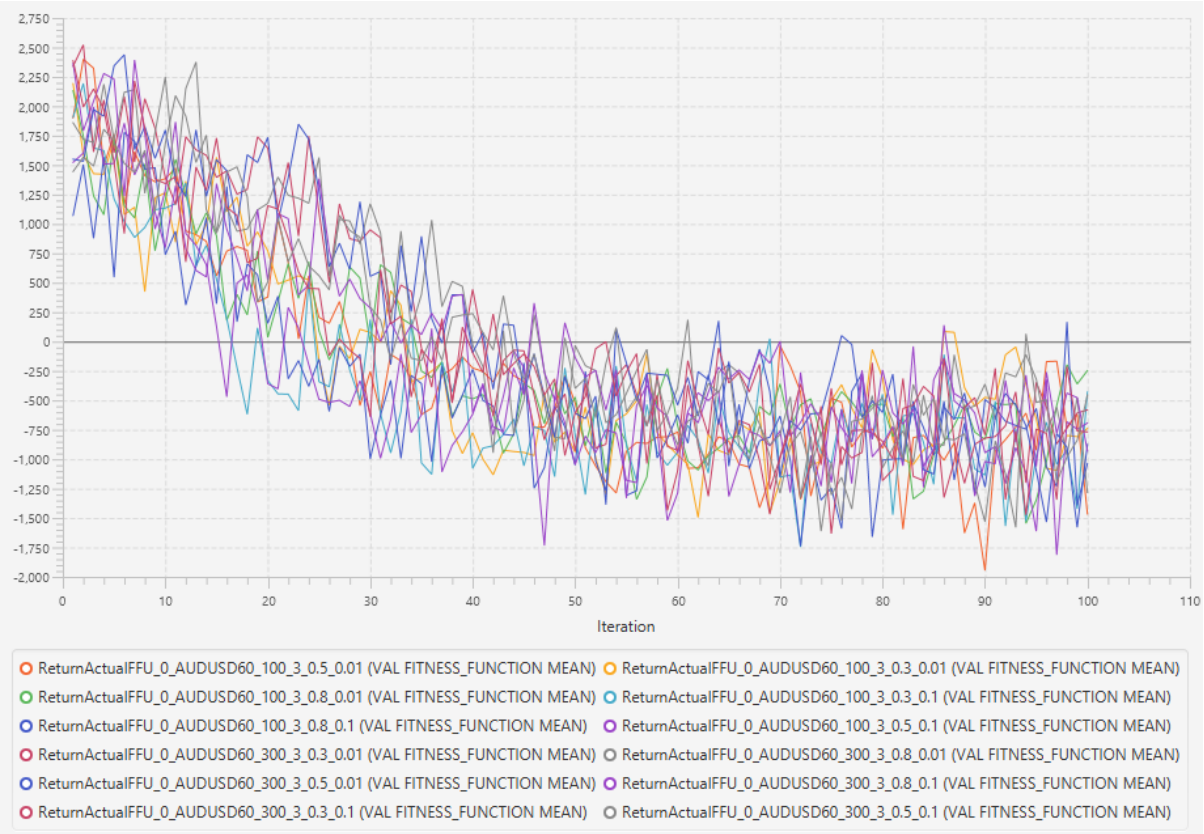


Figure 49: The average maximum return value at each iteration of the Genetic Algorithm using validation data

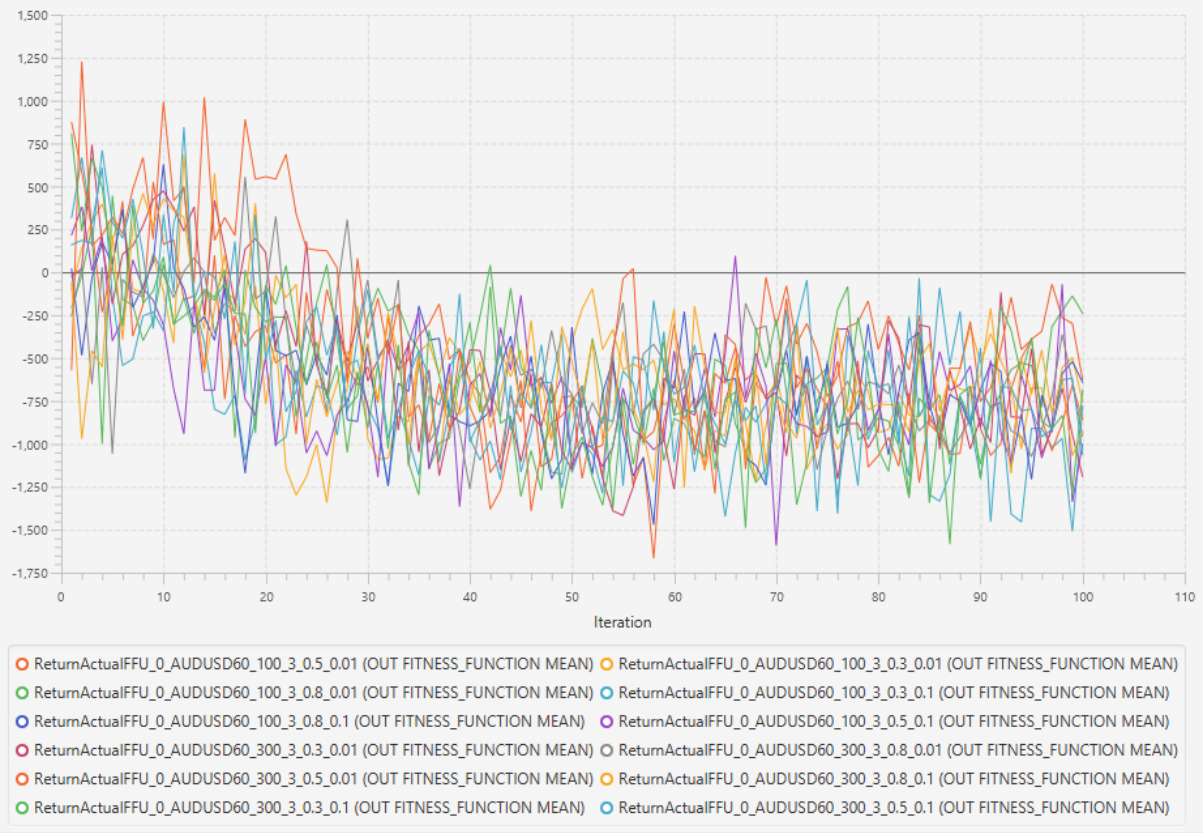


Figure 50: The average maximum return value at each iteration of the Genetic Algorithm using outsample

Figure 51 shows the mean (shown as a red line), maximum (green) and minimum (orange) return value of the population at each iteration of the Genetic Algorithm for a four separate Genetic Algorithm runs. Each graph is derived from insample data and all use different Genetic Algorithms parameter settings. The graphs show the population of trading strategies converging between 20 and 60 iterations of the Genetic Algorithm. The results were obtained from 1,200 Genetic Algorithm runs, and early convergence can be seen in the vast majority of Genetic Algorithm runs. The cause of this could be high selection pressure. Selection pressure is the tendency to select the fittest trading strategies for the crossover stage of the Genetic Algorithm. High selection pressure can reduce the genetic diversity of the population which leads to early convergence. A population of trading strategies is genetically diverse if it possesses a diverse set of technical analysis interpretations across the strategies. Selection pressure guides the search for optimal trading strategies, whilst genetic diversity ensures that the search space is adequately explored.

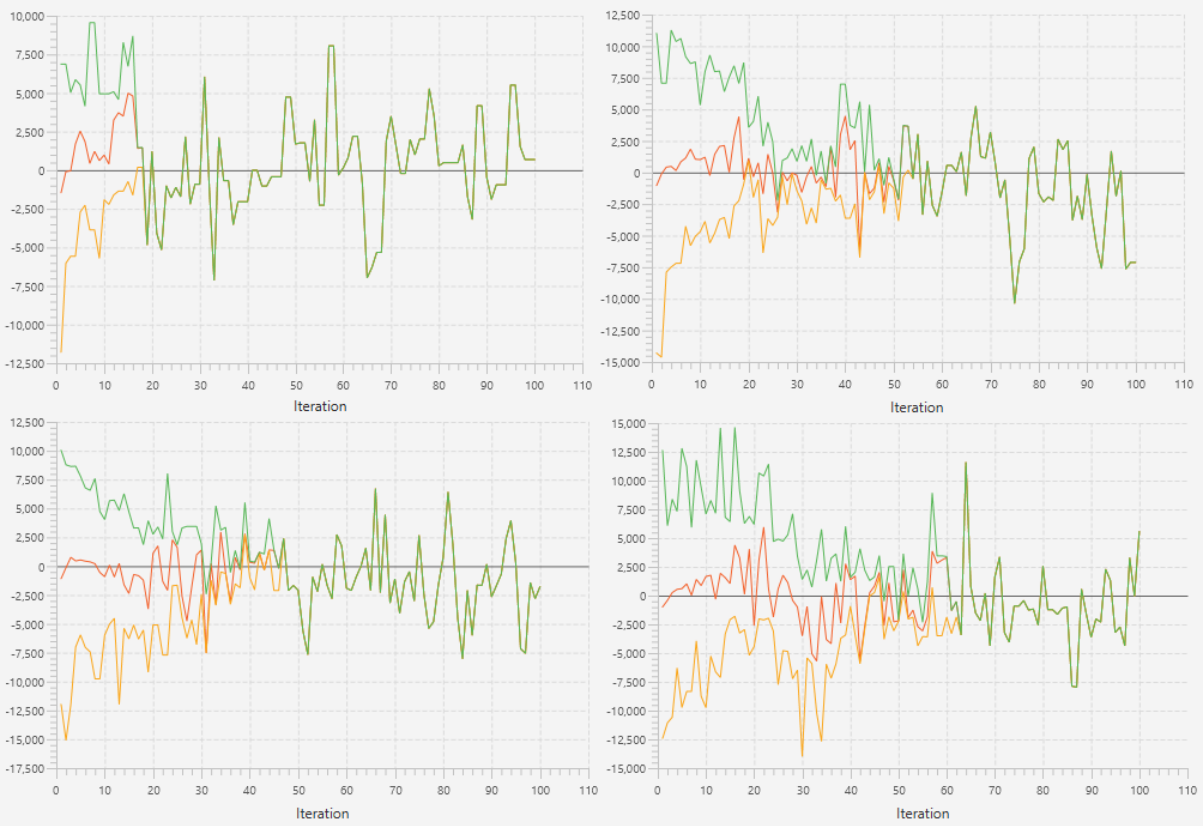


Figure 51: The mean (red), maximum (green) and minimum (orange) return values at each iteration of the Genetic Algorithm of four Genetic Algorithm runs

6.3.2 Tournament Selection

The previous Genetic Algorithm configurations did not increase in fitness before converging or decreasing in fitness. The fittest trading strategies during the initial iterations of the Genetic Algorithm seem to offer the best choice for picking profitable trading strategies compared to subsequent iterations. The results of the previous experiment converged early possibly due to high selection pressure. To test the assumption that high selection pressure is the problem, the selection technique used in the selection stage will be changed.

Figure 52 shows the average performance of 100 Genetic Algorithm runs of the fittest trading strategy individuals at each iteration of the Genetic Algorithm using the insample dataset. The Genetic Algorithm now uses the tournament selection technique instead of the universal sampling technique for selecting trading strategies during the selection stage of the Genetic Algorithm. The 16 different parameter setting configurations of the Genetic Algorithm that are explored are shown in Table 13. Figures 53 and 54 show the fittest

trading strategies using validation and outsample data. The average fitness values of the fittest trading strategies converge much faster than was the case when using the universal sampling selection technique. Inspecting individual Genetic Algorithm runs, the population generally converged within 5 to 30 iterations. This suggests that the Genetic Algorithm's tournament parameter settings may be causing early convergence with higher selection pressure.

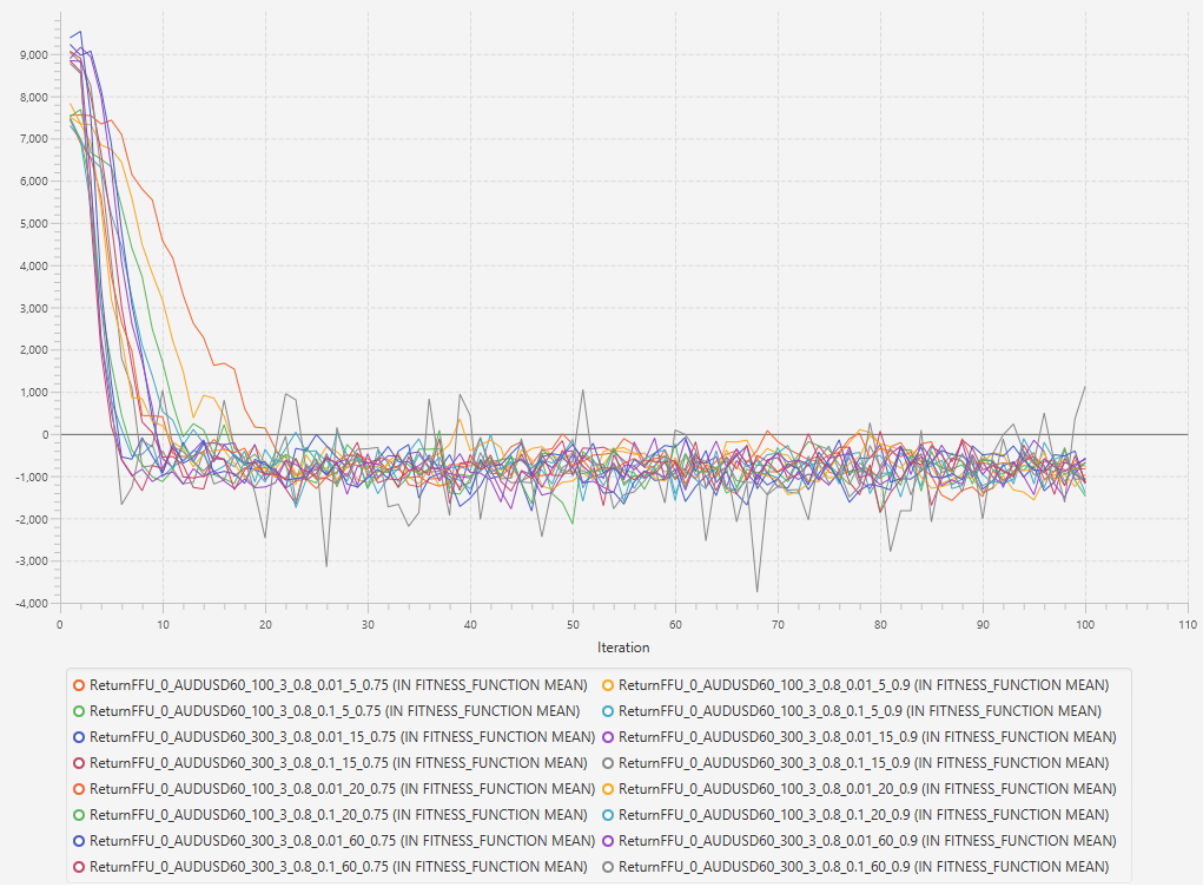


Figure 52: The average maximum return value at each iteration of the Genetic Algorithm using insample data

Stage	Parameter	Parameters explored
All	Size of technical analysis interpretation pool	10,000
All	Population size	100 and 300
All	Trading strategy size	3
Selection	Tournament win probability p	0.75 and 0.9
Selection	Tournament size k	5% and 20% of the population size parameter
Selection	Fitness function	Return
Crossover	Crossover probability	0.8
Mutation	Mutation rate	0.01 and 0.1

Table 13: Genetic Algorithm parameter settings

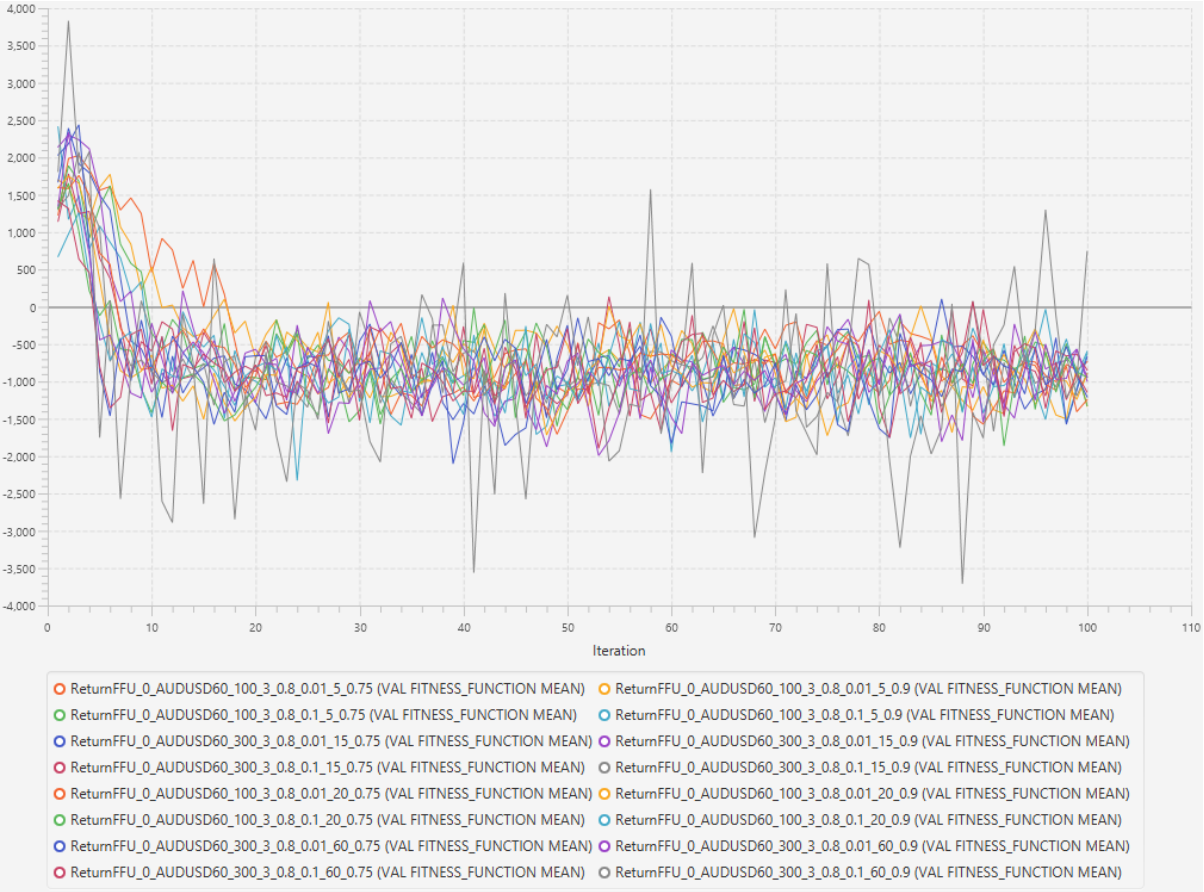


Figure 53: The average maximum return value at each iteration of the Genetic Algorithm using validation data

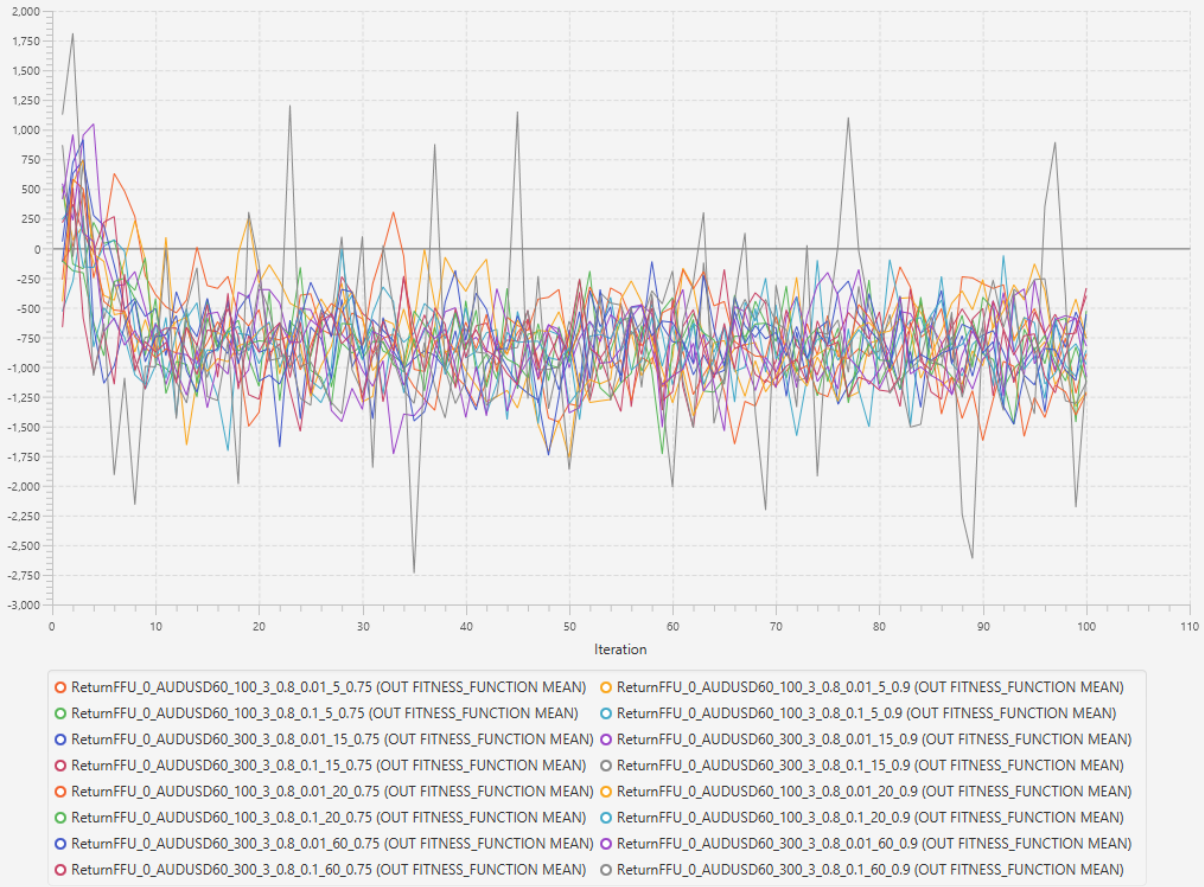


Figure 54: The average maximum return value at each iteration of the Genetic Algorithm using outsample data

To lower the selection pressure, the probability p of winning a tournament is reduced to the values 0.25 and 0.5. The 16 different parameter setting configurations of the Genetic Algorithm parameter settings are given in Table 14.

In Figure 55, two Genetic Algorithm parameter settings that have a population size of 100, tournament size of 5 trading strategies and probability p of being selected 0.25, did not converge within the 100 iterations of the Genetic Algorithm. The average return of the fittest trading strategies slowly decline.

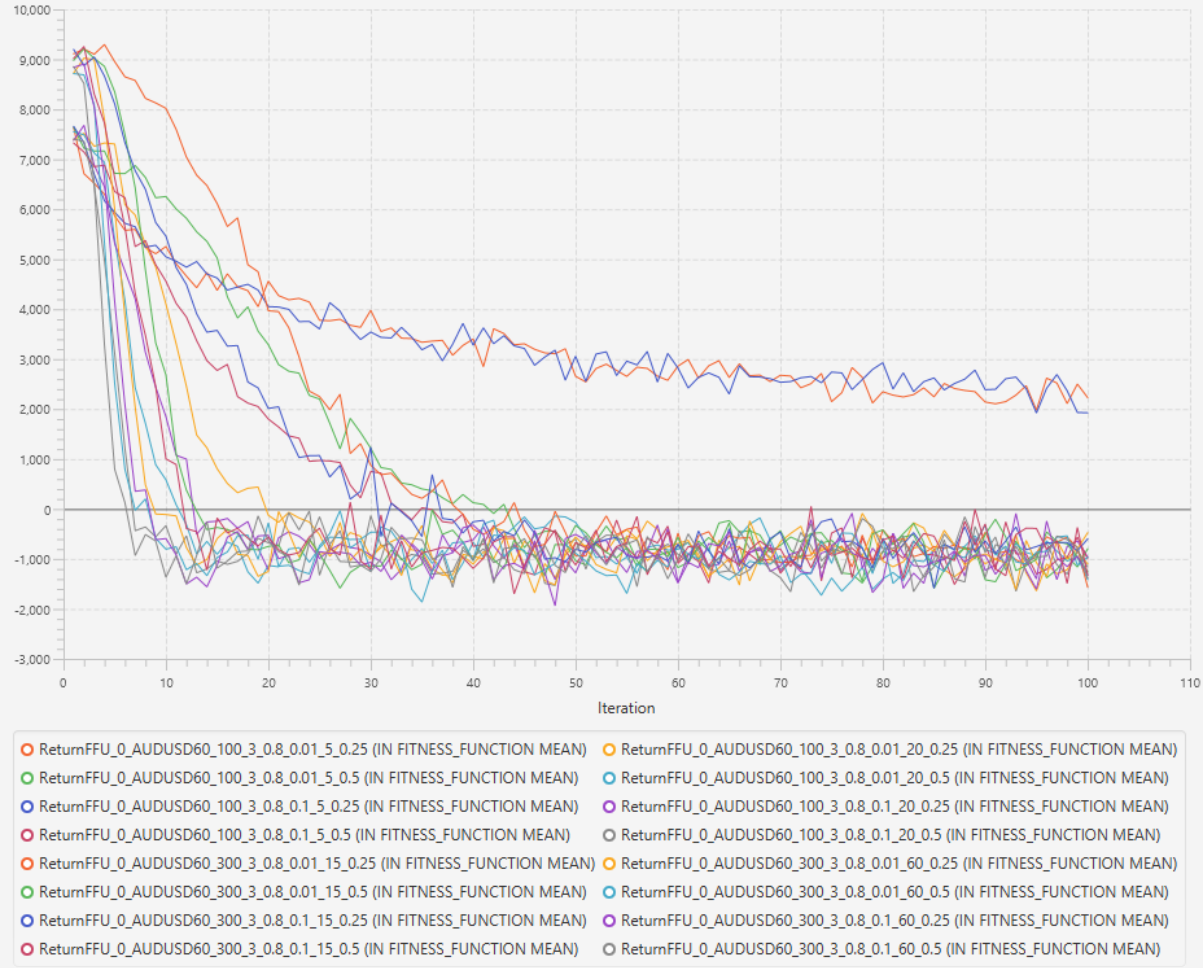


Figure 55: The average maximum return value at each iteration of the Genetic Algorithm using insample data

Stage	Parameter	Parameters explored
All	Size of technical analysis interpretation pool	10,000
All	Population size	100 and 300
All	Trading strategy size	3
Selection	Tournament win probability p	0.25 and 0.5
Selection	Tournament size k	5% and 20% of the population size parameter
Selection	Fitness function	Return
Crossover	Crossover probability	0.8
Mutation	Mutation rate	0.01 and 0.1

Table 14: Genetic Algorithm parameter settings

Figure 56 shows the average return of 100 Genetic Algorithm runs of the fittest trading strategies obtained from insample data on validation data. The two Genetic Algorithm

parameter settings that slowly decline on insample data at each iteration of the Genetic Algorithm also show some profitable average return values. This result would be beneficial if the validation dataset were to be considered as a form of out of sample dataset. Figure 57 however shows average return values on the outsample dataset are all below zero for all the parameter settings explored.

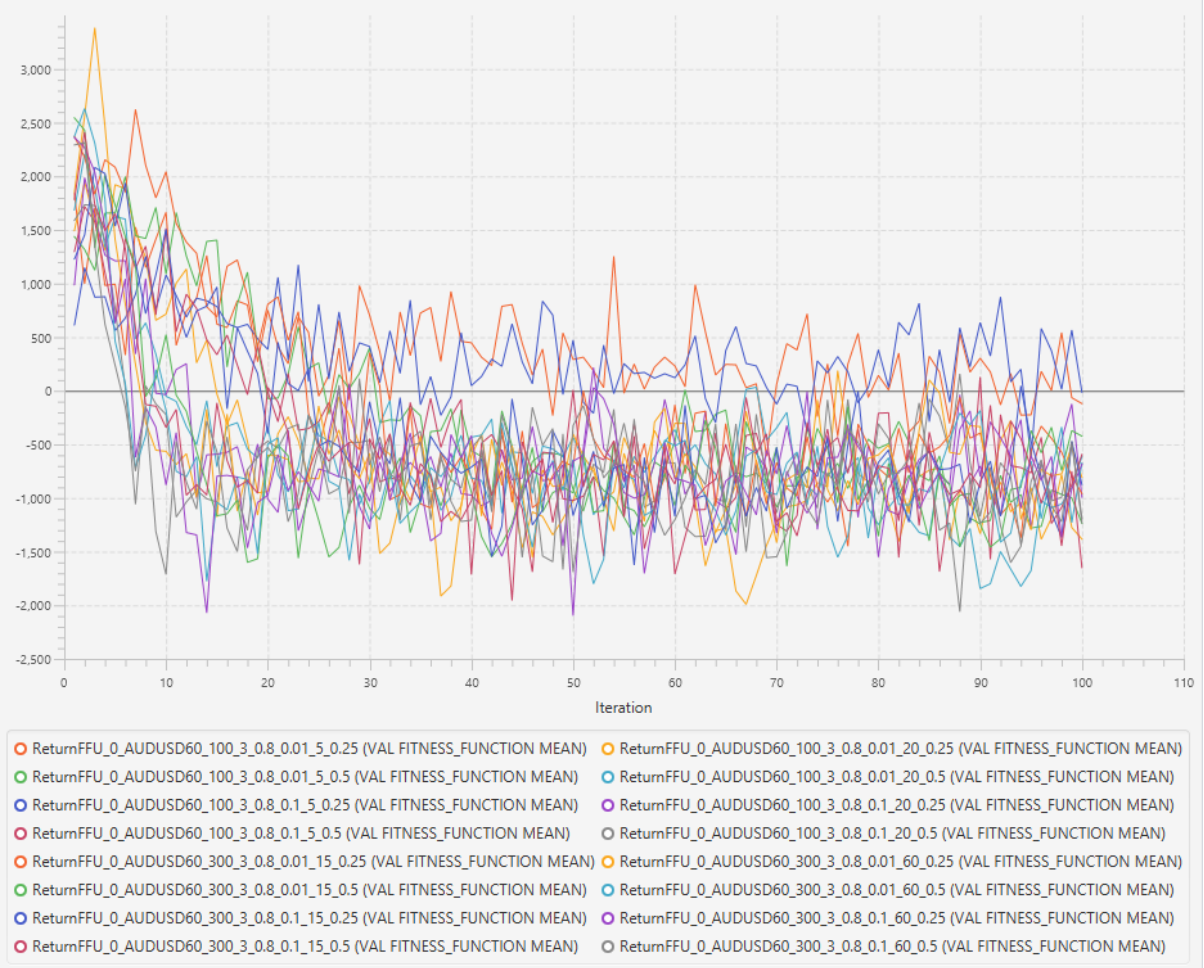


Figure 56: The average maximum return value at each iteration of the Genetic Algorithm using validation data

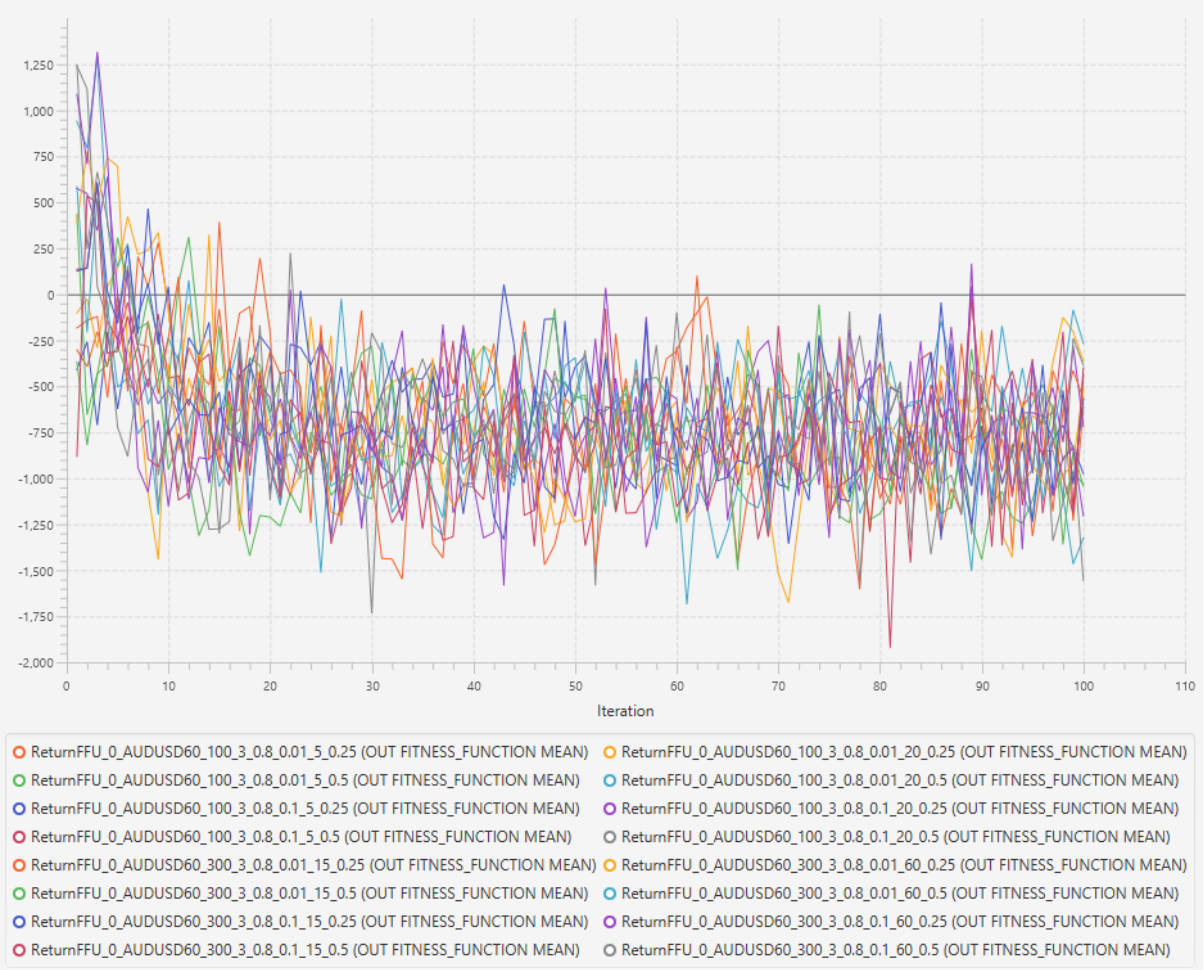


Figure 57: The average maximum return value at each iteration of the Genetic Algorithm using outsample data

The Genetic Algorithm configurations used in the last two experiments show early convergence for the majority of parameter settings. The average return of two Genetic Algorithm parameter settings did not permanently drop below zero return on insample and validation data, however all Genetic Algorithm configurations led to negative average return on outsample data.

The fitness at each iteration of the Genetic Algorithm should increase before decreasing or converging. None of the Genetic Algorithm configurations were able to improve the initial population of trading strategies. Changing the selection process from five point universal sampling to tournament selection did not solve the perceived early convergence problem. The trading strategies used in previous experiments each contain 3 technical analysis interpretations and use a majority vote decision. Changing a technical analysis interpretation in a trading strategy may drastically change the trading strategy's

performance and trading strategies with more technical analysis interpretations will be explored in the following section.

6.3.3 Increasing Trading Strategy Size

The crossover stage of the Genetic Algorithm involves swapping a random technical analysis interpretation of one trading strategy with another random technical analysis interpretation of another trading strategy. The mutation stage of the Genetic Algorithm involves randomly changing a technical analysis interpretation with another random technical analysis interpretation. The trading strategies used in the previous Genetic Algorithms contain 3 technical analysis interpretations per trading strategy. A good combination of two technical analysis interpretations within the trading strategy may not be realised by the fitness function as the third technical analysis interpretation can have a large impact on the trading strategy's buy and sell decisions. A dramatic change in a trading strategy's buy and sell decisions can completely alter its fitness value.

Increasing the number of technical analysis interpretations in a trading strategy may stabilise the trading strategy's fitness value as the trading strategy undergoes the evolutionary process. The buy and sell decisions and fitness value of a trading strategy containing 5 technical analysis interpretations in majority vote that contains a good combination of 3 or 4 technical analysis interpretations is unlikely to be severely affected by the replacement of a technical analysis interpretation outside the good combination of technical analysis interpretations within the trading strategy. Increasing the number of technical analysis interpretations per trading strategy should therefore reduce the chance of dramatic changes in buy and sell decisions, helping to facilitate the local search process of the Genetic Algorithm.

Figure 58 shows the average performance of 100 Genetic Algorithm runs of the fittest trading strategy individuals at each iteration of the Genetic Algorithm using the insample dataset. Figures 59 and 60 show the fittest trading strategies using validation and outsample data. The figures show the results from trading strategies containing 5 technical analysis interpretations; further results obtained from trading strategies consisting of 6 and 7 technical analysis interpretations are reported in Appendix A. The 72 different parameter setting configurations of the Genetic Algorithm that are explored are shown in Table 15.

Despite expectations, the results of the Genetic Algorithms using trading strategies with 5, 6 and 7 technical analysis interpretations all follow the same pattern as the results from previous sections.

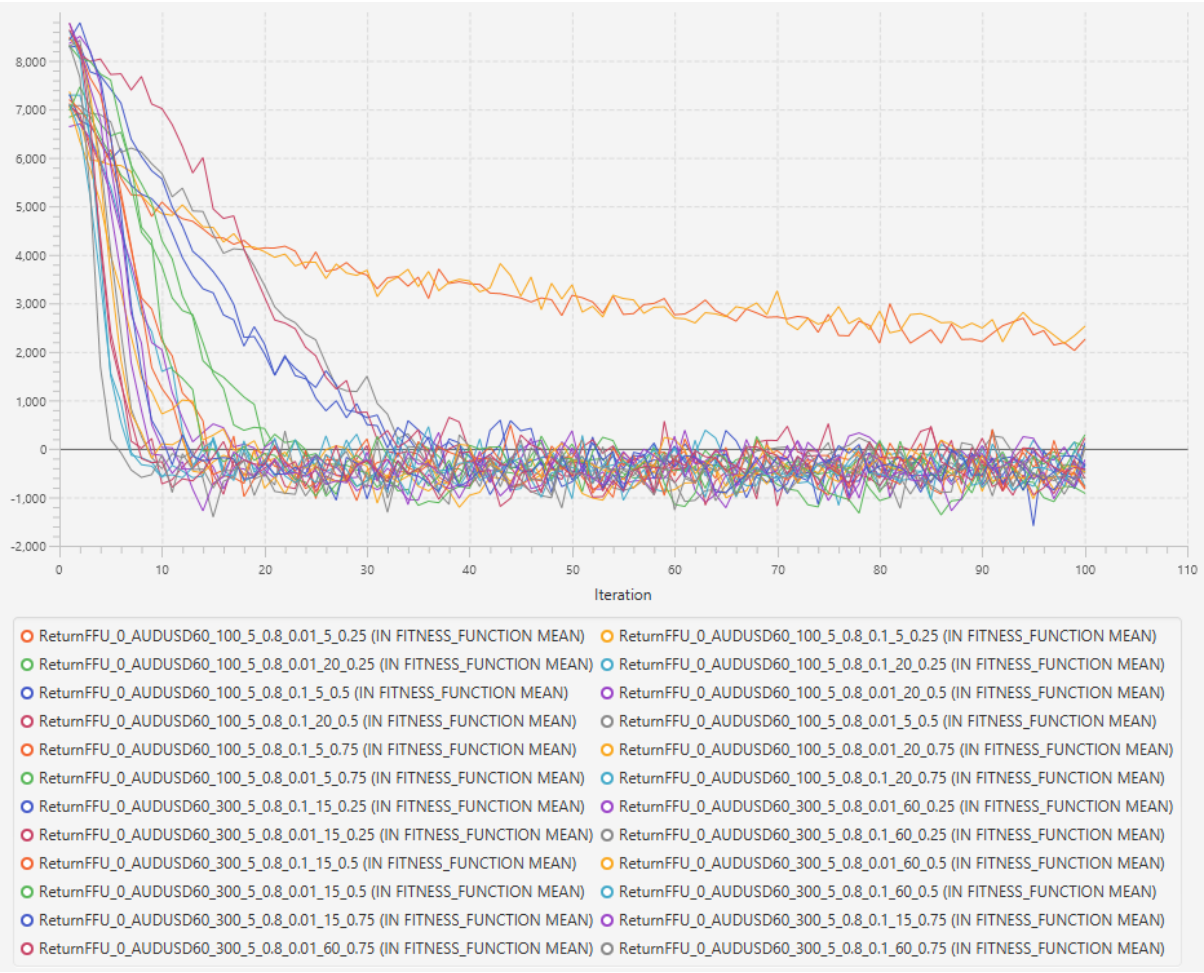


Figure 58: The average maximum return value at each iteration of the Genetic Algorithm using insample data

Stage	Parameter	Parameters explored
All	Size of technical analysis interpretation pool	10,000
All	Population size	100 and 300
All	Trading strategy size	5, 6 and 7
Selection	Tournament win probability p	0.25, 0.5 and 0.75
Selection	Tournament size k	5% and 20% of the population size parameter
Selection	Fitness function	Return
Crossover	Crossover probability	0.8
Mutation	Mutation rate	0.01 and 0.1

Table 15: Genetic Algorithm parameter settings

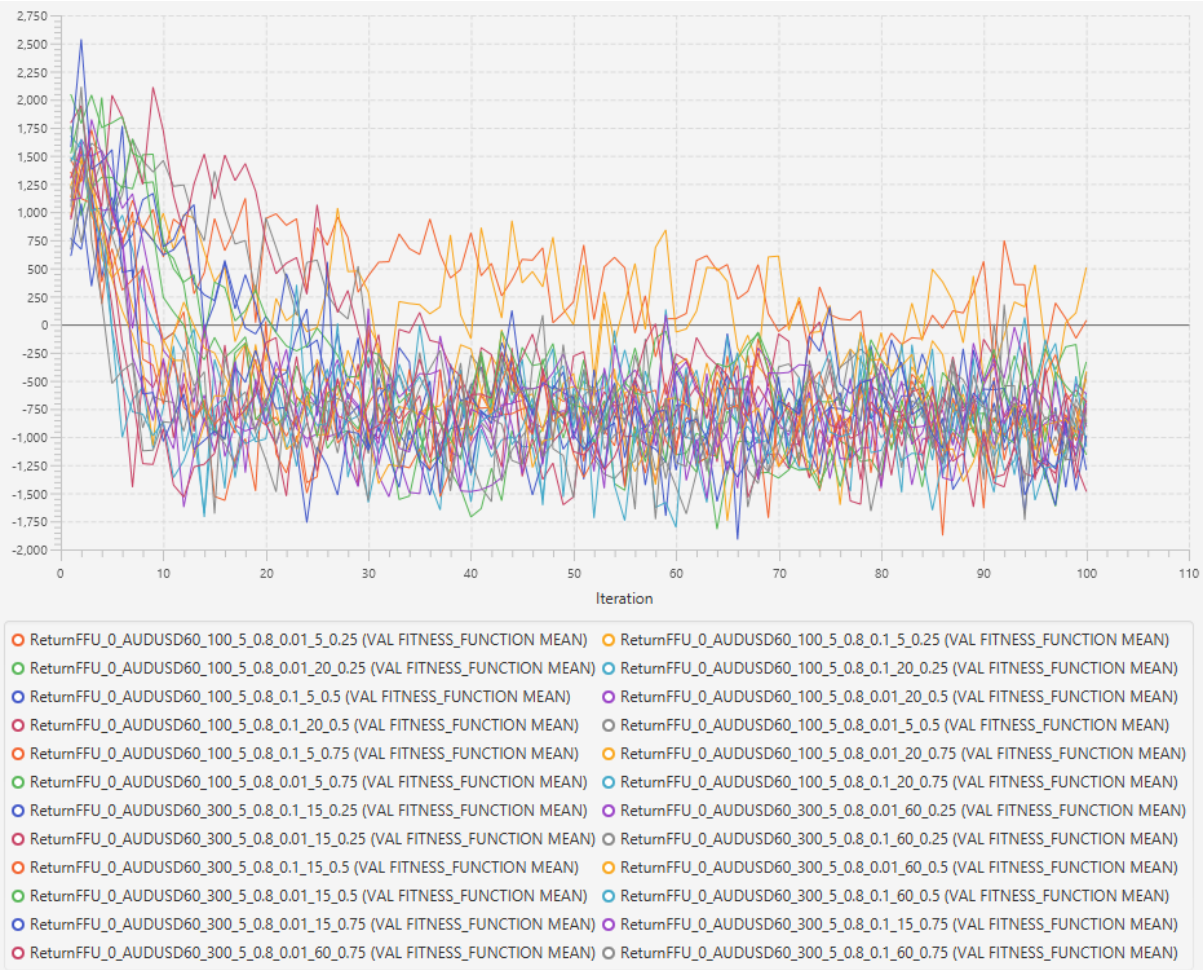


Figure 59: The average maximum return value at each iteration of the Genetic Algorithm using validation data

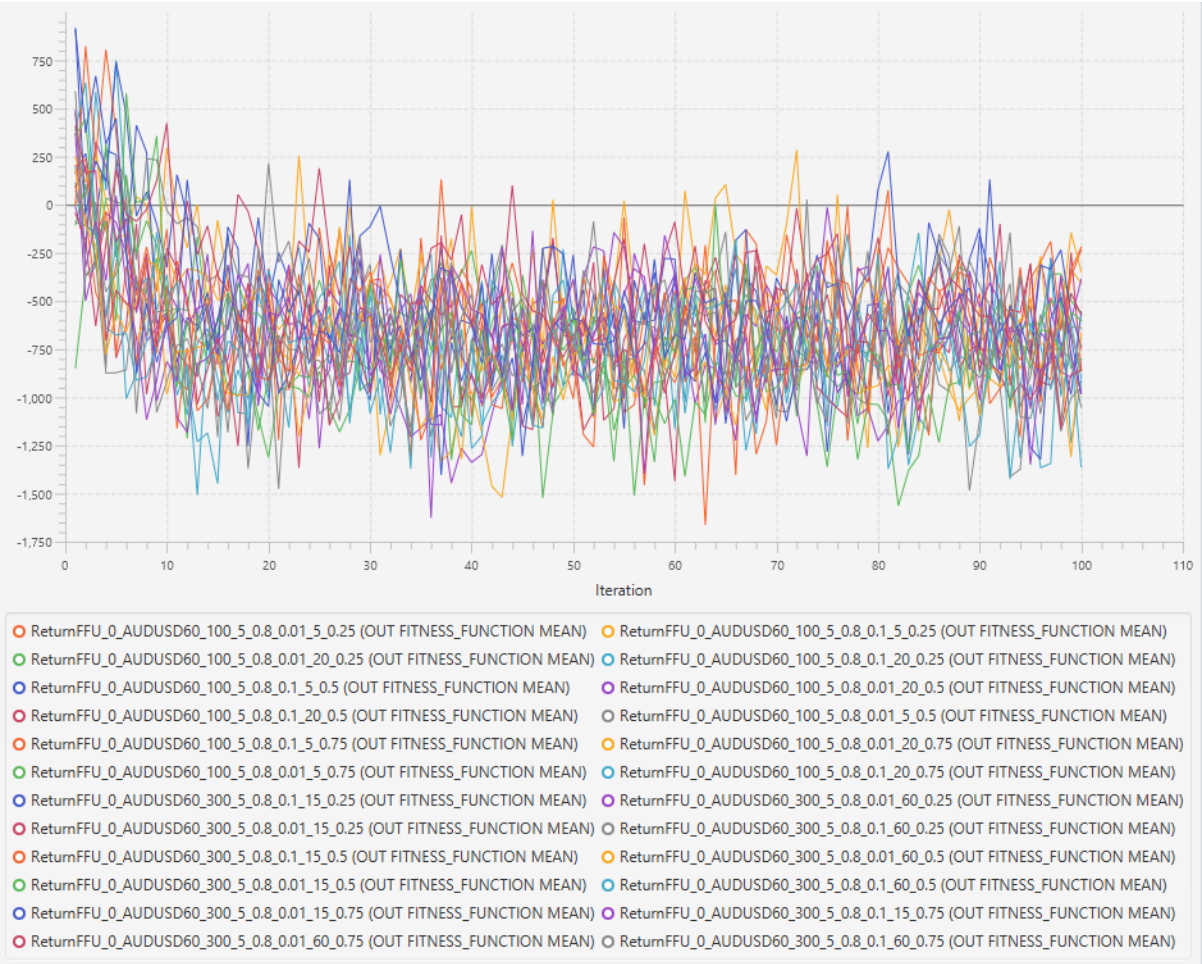


Figure 60: The average maximum return value at each iteration of the Genetic Algorithm using outsample data

Similar to the results in the previous section, two Genetic Algorithm parameter settings that have a population size of 100, tournament size of 5 trading strategies and probability p of being selected 0.25, did not converge within the 100 iterations of the Genetic Algorithm, see Figure 61.

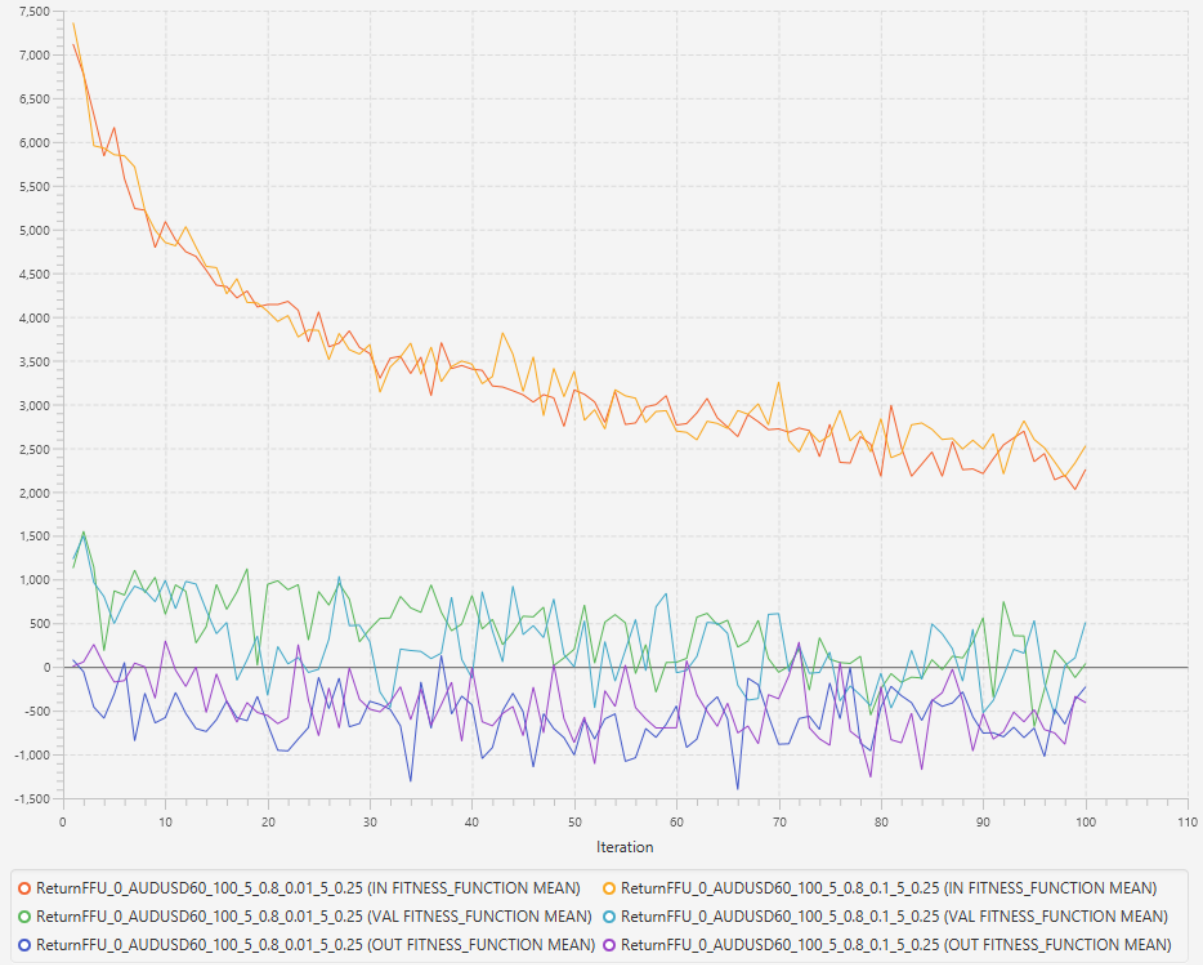


Figure 61: Results of two Genetic Algorithm configurations that did not converge within 100 iterations of the Genetic Algorithm

The Genetic Algorithm configurations used in the previous experiments show early convergence for the majority of parameter settings. The average return of two Genetic Algorithm parameter settings did not permanently drop below zero return on insample and validation data, however all Genetic Algorithm configurations led to negative average return on outsample data.

Increasing the number of technical analysis interpretations per trading strategy did not resolve the perceived problem of early convergence. So far none of the Genetic Algorithm configurations were able to improve the initial population of trading strategies. The next section will explore whether the population size is causing the perceived early convergence.

6.3.4 Increasing Population Size

The experiments involving Genetic Algorithms so far are failing to improve on the fitness of the fittest trading strategies within the population, and so the population converges rather

quickly. Increasing the population size might potentially reduce the chance of premature convergence and so could have a positive effect on the population's fitness results. Increasing the population size however increases the runtime of the Genetic Algorithm.

Figure 62 shows the average performance of 100 Genetic Algorithm runs of the fittest trading strategy individuals at each iteration of the Genetic Algorithm using the insample dataset. The Genetic Algorithm again uses the tournament selection technique instead of the universal sampling technique for selecting trading strategies for the crossover stage. The 16 different parameter setting configurations of the Genetic Algorithm that are explored are shown in Table 16. Figures 63 and 64 show the fittest trading strategies using validation and outsample data. The average fitness values of the fittest trading strategies again converge quickly on insample, validation and outsample data. Inspecting individual Genetic Algorithm runs, the population generally converged within 10 iterations.

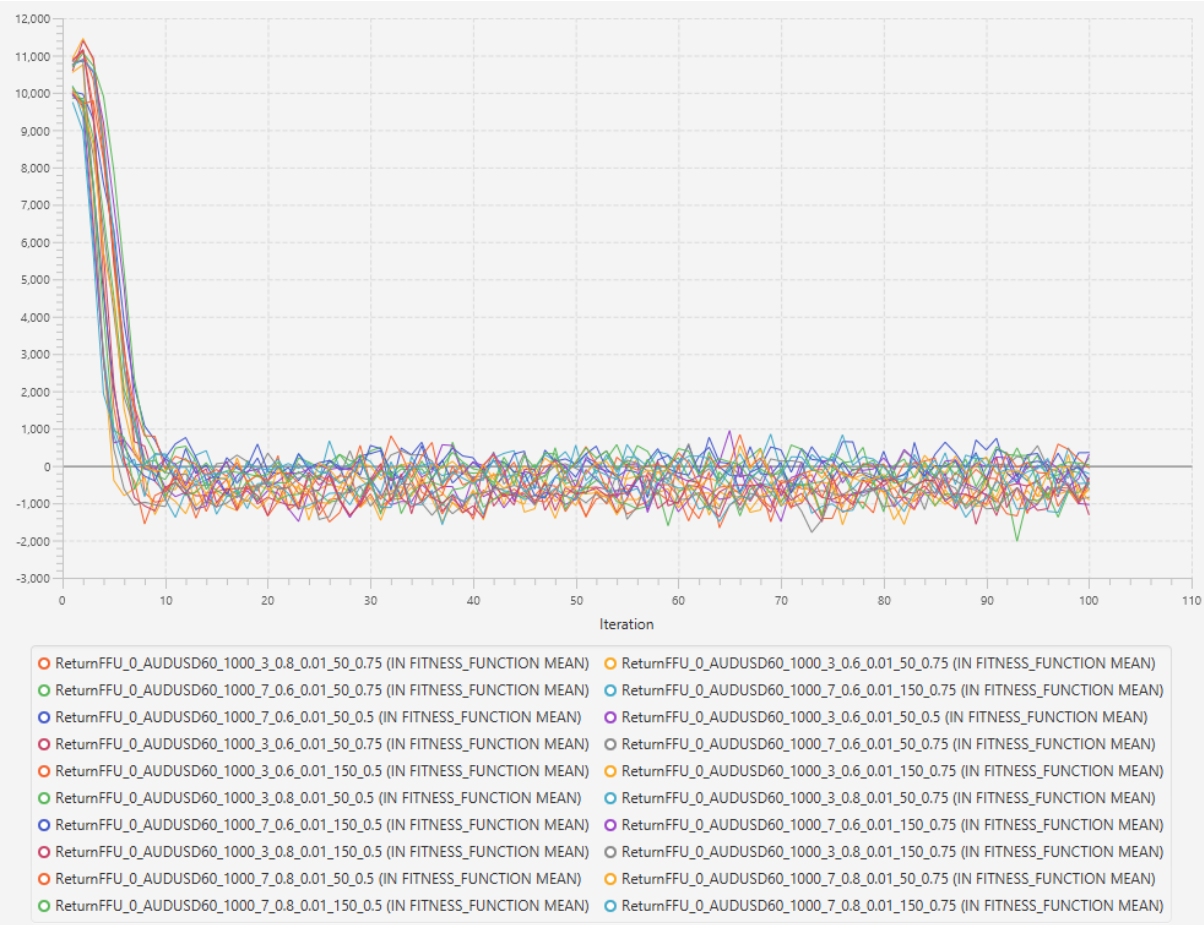


Figure 62: The average maximum return value at each iteration of the Genetic Algorithm using insample data

Stage	Parameter	Parameters explored
All	Size of technical analysis interpretation pool	10,000
All	Population size	1000
All	Trading strategy size	3 and 7
Selection	Tournament win probability p	0.5 and 0.75
Selection	Tournament size k	5% and 20% of the population size parameter
Selection	Fitness function	Return
Crossover	Crossover probability	0.6 and 0.8
Mutation	Mutation rate	0.01

Table 16: Genetic Algorithm parameter settings

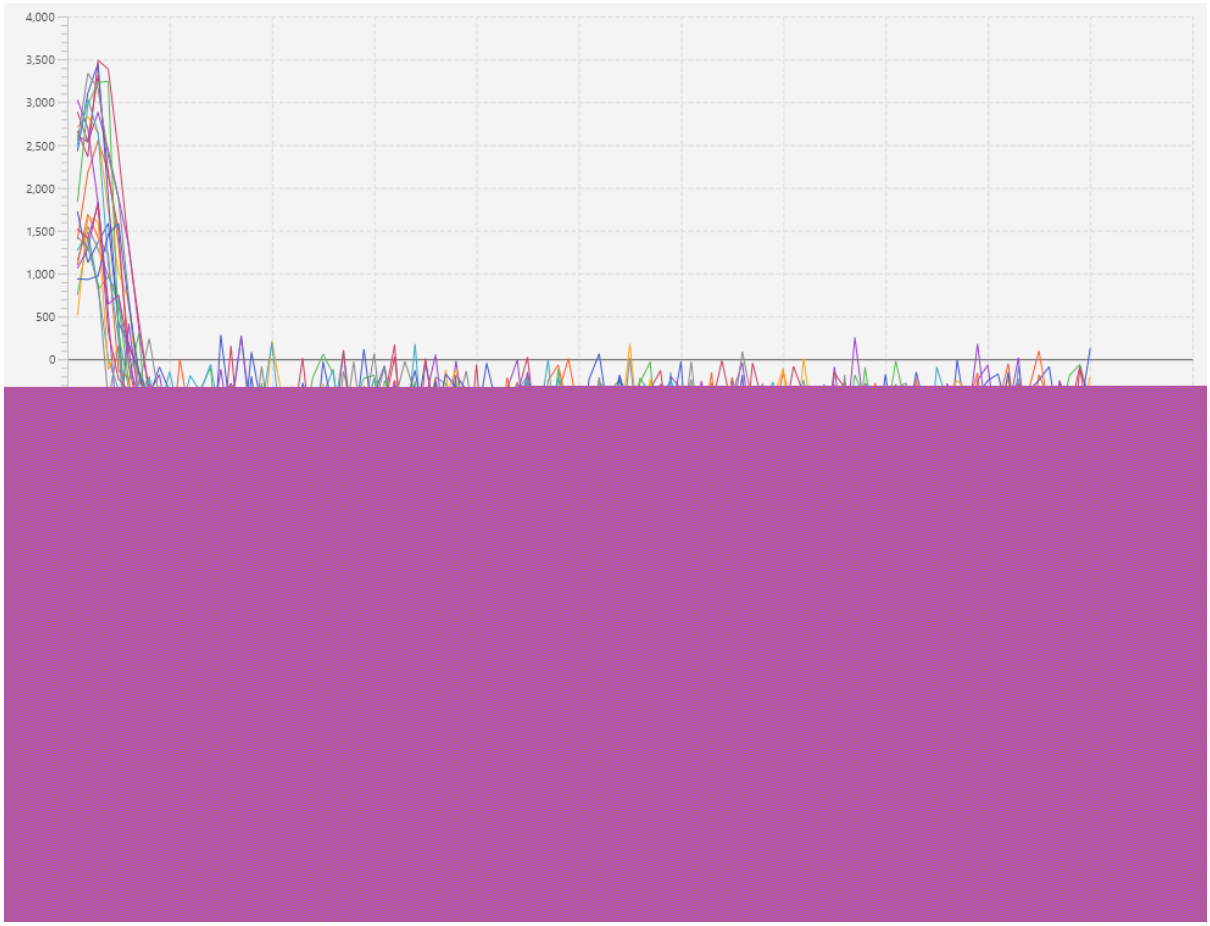


Figure 63: The average maximum return value at each iteration of the Genetic Algorithm using validation data

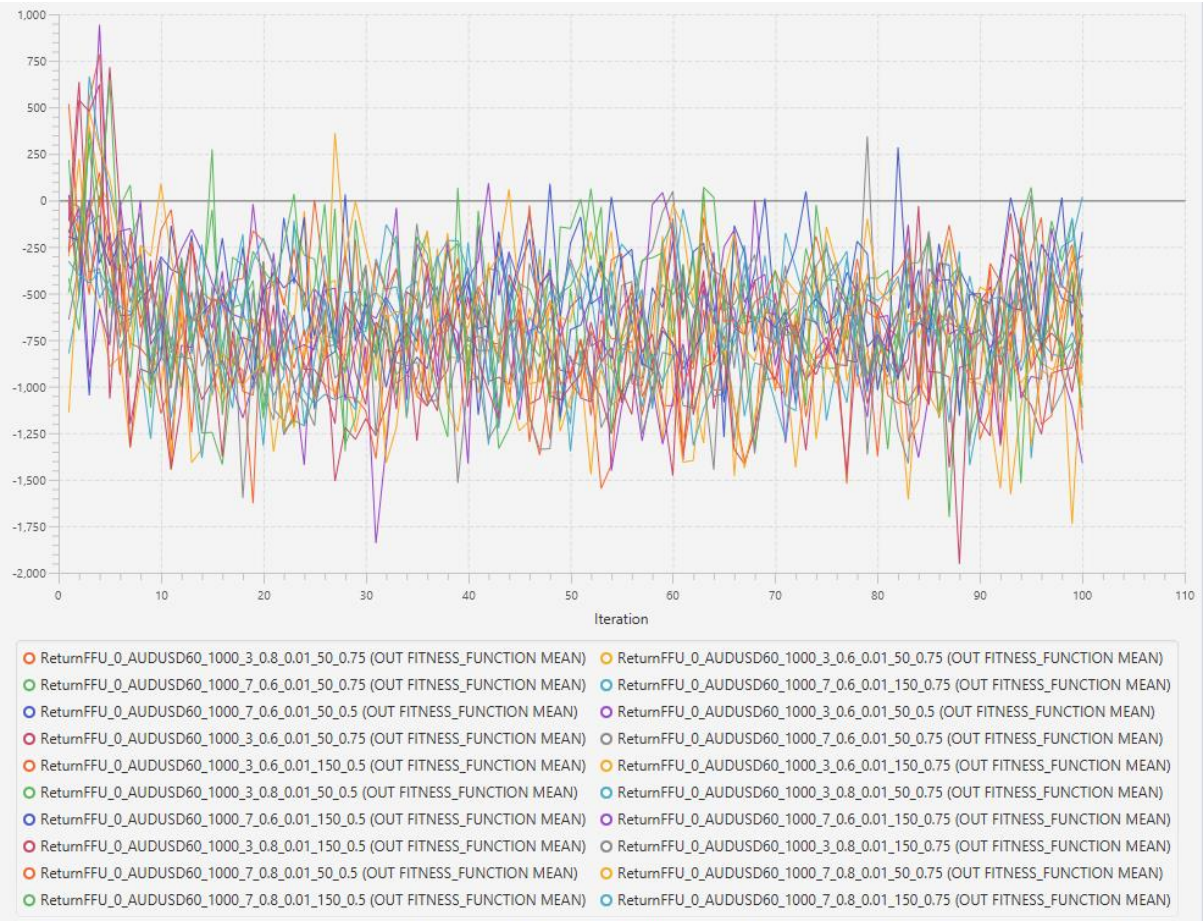


Figure 64: The average maximum return value at each iteration of the Genetic Algorithm using outsample data

Increasing the population size from 100 and 300 to 10,000 trading strategies did not resolve the perceived problem of early convergence.

6.4 Conclusions

The experiments in this chapter explored different parameter settings of the Genetic Algorithm, increased the number of technical analysis interpretations per trading strategy, increased the population size and explored two different Genetic Algorithm selection techniques. The fitness values of the experiments converged quickly and the Genetic Algorithm did not improve the population's return fitness function. The fittest trading strategies at each iteration of the Genetic Algorithm using insample data did not increase either. As previously mentioned, the fitness value of the fittest trading strategy in the population should increase at each iteration of the Genetic Algorithm before converging because it is only by fluke chance the population starts with an optimal trading strategy. The

fitness values of the trading strategies using validation data drops within the first few iterations in a similar way as for the trading strategies using insample data drop.

Two Genetic Algorithm configurations that used a population size of 100, tournament size of 5 trading strategies and probability p of being selected 0.25, did not converge within the 100 iterations of the Genetic Algorithm. Though the population did not converge, the fitness of the fittest trading strategy in the population decreased with each iteration on insample data.

The difference between a trading strategy's fitness value and its neighbouring trading strategies' fitness values may be too drastic for the Genetic Algorithm. Neighbouring trading strategies are produced by changing a single technical analysis interpretation in a trading strategy. The next chapter investigates the landscape of the search space of the Genetic Algorithm and how the trading strategies are changed between each iteration of the Genetic Algorithm. In attempt to create a Genetic Algorithm that succeeds in evolving populations for trading strategies to higher fitness states. The next chapter will also attempt to fix the Genetic Algorithm by exploring various fitness functions, exploring different market datasets and reducing the pool of technical analysis interpretations.

Chapter 7 – Further Investigation into the Generation of Trading Strategies using Genetic Algorithms

The previous chapter described an attempt to produce profitable trading strategies using the Genetic Algorithm. Various Genetic Algorithm configurations were explored, but these configurations failed to create profitable trading strategies. The fitness of the populations for the tested Genetic Algorithm configurations converged within a few iterations, and the fitness of the initial population decreased from the outset on insample data.

This chapter extends the previous chapter, attempting to apply Genetic Algorithms to create 'good' trading strategies by maximising the value of different performance metrics. Traders in a similar way attempt to find 'good' trading strategies by also maximising and minimising the values of desirable performance metrics. This chapter also investigates whether the failure to produce 'good' trading strategies originates from the magnitude of the differences between the fitness values of neighbouring trading strategies. In other words, the landscape of the search space being searched for local optima is very uneven.

7.1 Investigating the Genetic Algorithm's Local Search

The Genetic Algorithm attempts to evolve a population of trading strategies by maximising some fitness function. The results from the previous chapter showed that the average fitness value of the fittest trading strategies on insample data did not increase when maximising the return performance metric. The Genetic Algorithm selects trading strategies at random with a bias towards picking the fittest trading strategies to breed during the selection process. The selection process allows the same trading strategy to be selected more than once. The fittest trading strategy is likely to breed multiple times which propagates the trading strategy's good technical analysis interpretation combinations throughout the population. When the trading strategies are bred, good combinations of technical analysis interpretations are combined with other technical analysis interpretations to form new trading strategies.

In the previous chapter the number of technical analysis interpretations per trading strategy was increased to try to reduce the size of a change in a trading strategy's fitness value resulting from the change of a single technical analysis interpretation. In other words, to produce a search space that is more even. This can happen during mutation when a

technical analysis interpretation is replaced or when a technical analysis interpretation is swapped during the crossover stage of the Genetic Algorithm. There was no difference in the Genetic Algorithm's results between using trading strategies made up of seven technical analysis interpretations when compared to ones constructed from three technical analysis interpretations.

To investigate whether the fitness value of a trading strategy is significantly affected by a single technical analysis interpretation replacement, the differences between the fitness value of the trading strategy and those of the neighbouring trading strategies (which are trading strategies obtained by replacing one of its technical analysis interpretations) need to be explored. Figure 65 shows the frequency distribution of the absolute differences in return values of neighbouring trading strategies on AUDUSD insample data. In total 10,000 technical analysis interpretations were randomly created and 10,000 trading strategies are created using the technical analysis interpretations. To obtain neighbouring trading strategies, a technical analysis interpretation in the trading strategy is replaced by another random technical analysis interpretation; this is done 100 times for each technical analysis interpretation in the trading strategy. Each trading strategy and neighbouring trading strategy contains 3 technical analysis interpretations and none are given duplicate technical analysis interpretations. In total 3,000,000 differences in return of trading strategies and neighbouring trading strategies were calculated.

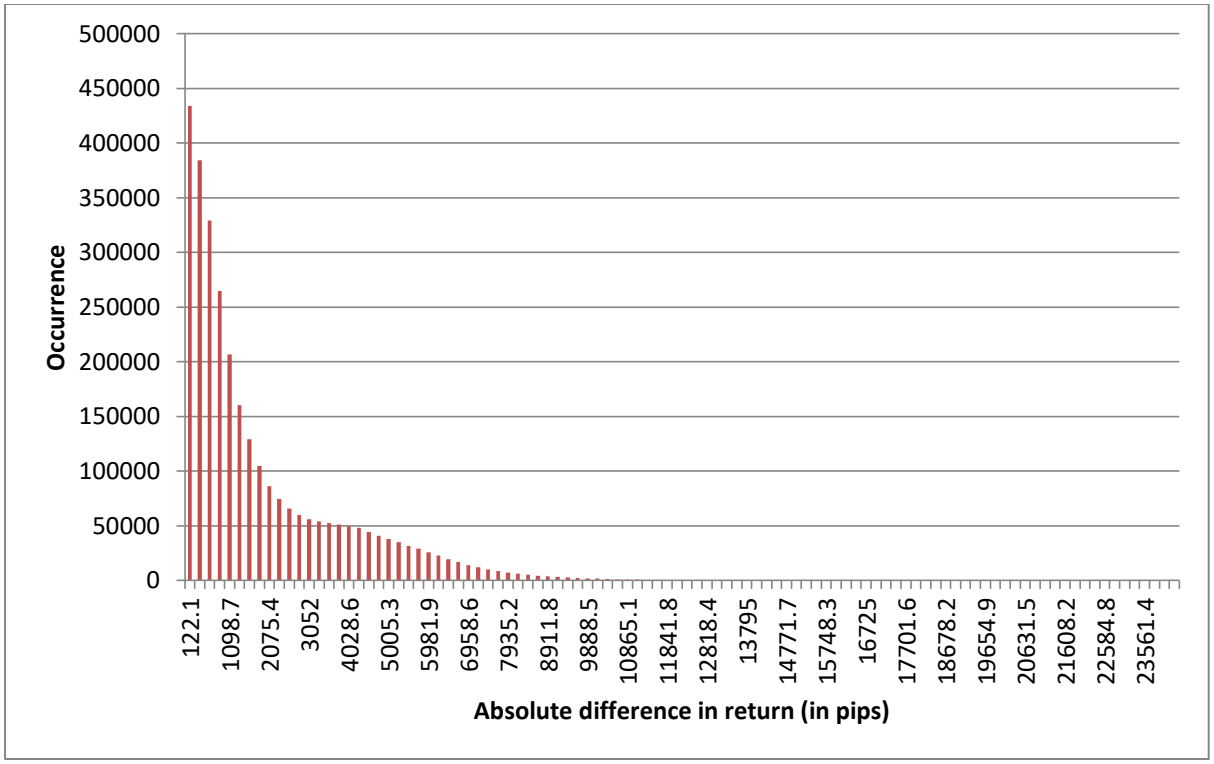


Figure 65: Histogram of 100 bins showing the absolute difference in return in pips of neighbouring trading strategies of size 3.

The mean absolute difference between a trading strategy's return value and neighbouring trading strategies is 1,971 pips, the median absolute difference is 1,096 pips and the standard deviation of the absolute difference is 2,151 pips. A pip is the smallest price move that a given exchange rate makes, it is a practical difference of a price, conventionally currency pairs are priced to four decimal places and the smallest change is 1/100 of 1%, or one basis point. Pips are being used here as a proxy for absolute return. Neighbouring trading strategies have an average of 1,971 pips difference in return but with the high standard deviation of 2,151 pips, the difference in return is unreliable.

The mean is easily influenced by outliers. Where, as here, there is a considerable difference of 44% between the median and mean. The distribution is highly skewed and the median is a better measure of central tendency. However, the standard deviation is much greater than the median of 1,096. Given that small changes in the trading strategy are not guaranteed to produce small changes in return, it is possible that the Genetic Algorithm is frequently performing a random search instead of moving to local optimal solutions

7.2 Alternative Fitness Functions

Previous experiments used the return performance metric of a trading strategy as the fitness function of the Genetic Algorithm. Other performance metrics may be better suited as fitness functions and may evolve 'good' trading strategies. Similar to the experiments in Chapter 6, this section uses the same AUDUSD hourly data segments for insample, validation and outsample datasets and runs the Genetic Algorithm 100 times to produce average performance results from the fittest trading strategies in the population at each iteration of the algorithm. The section explores the use of different fitness functions, where each Genetic Algorithm employs a single metric. The performance metrics used are the Sharpe ratio, Sortino ratio and trade success rate performance metrics. These performance metrics were chosen because the Sharpe ratio and Sortino ratio are used often in the literature and these ratios guide the Genetic Algorithm to create good return per unit risk trading strategies which are two desired qualities of a 'good' trading strategy. A trading strategy with a high trade success rate performance metric is also a desirable quality of a 'good' trading strategy.

The 24 different parameter setting configurations of the Genetic Algorithm are shown in Table 17. As well as the fitness function, the parameter settings of the Genetic Algorithm are also uncertain. In order to limit the number of time-consuming experiments, a limited range of values centring on crossover = 0.8 and mutation rate = 0.01 are used. These values are typical in the literature (Berutich, López, Luna, & Quintana, 2016; Pinto, Neves, & Horta, 2015; Wang, An, Liu, & Huang, 2016).

Stage	Parameter	Parameters explored
All	Size of technical analysis interpretation pool	10,000
All	Population size	300
All	Trading strategy size	3 and 7
Selection	Tournament win probability p	0.5 and 0.75
Selection	Tournament size k	5% and 20% of the population size parameter
Selection	Fitness function	Sharpe ratio, Sortino ratio and Trade success rate
Crossover	Crossover probability	0.8
Mutation	Mutation rate	0.01

Table 17: Genetic Algorithm parameter settings

7.2.1 Sharpe Ratio Fitness Function

Figure 66 shows the average performance of the fittest trading strategy individuals which maximise the Sharpe ratio performance metric at each iteration of the Genetic Algorithm using the insample dataset. Figures 67 and 68 show the fittest trading strategies using validation and outsample data. Similar to the results of the return fitness function, the maximum fitness in the population dropped from the initial value and the values converged within 15 iterations.

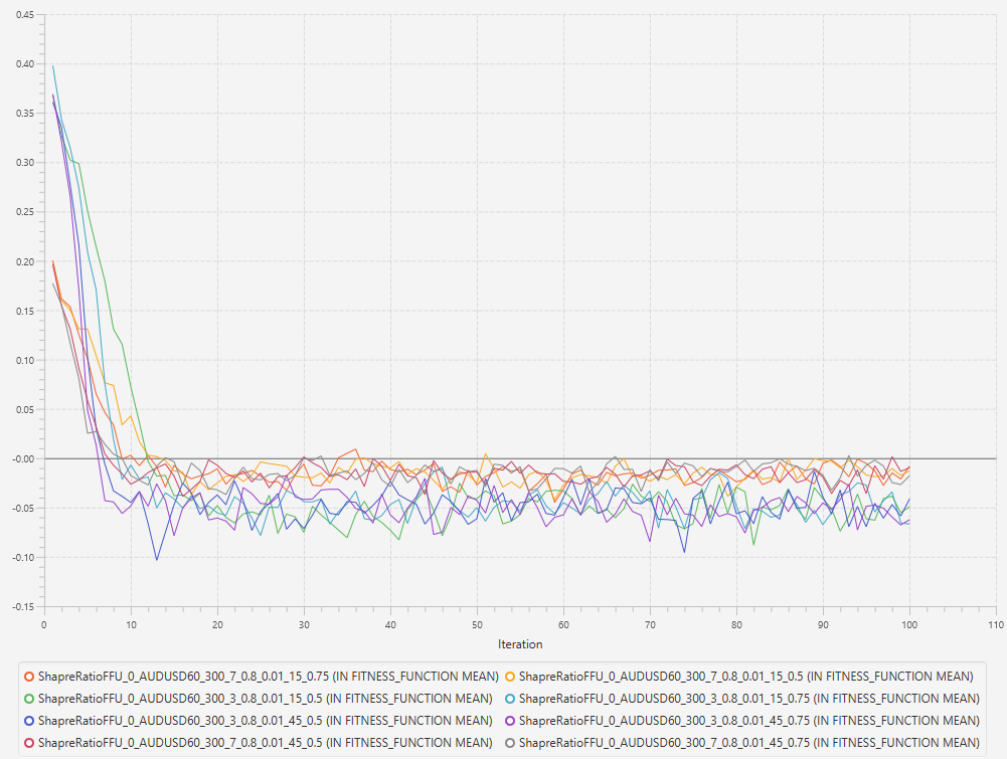


Figure 66: The average maximum Sharpe ratio value at each iteration of the Genetic Algorithm using insample data. Vertical axis is fitness function value

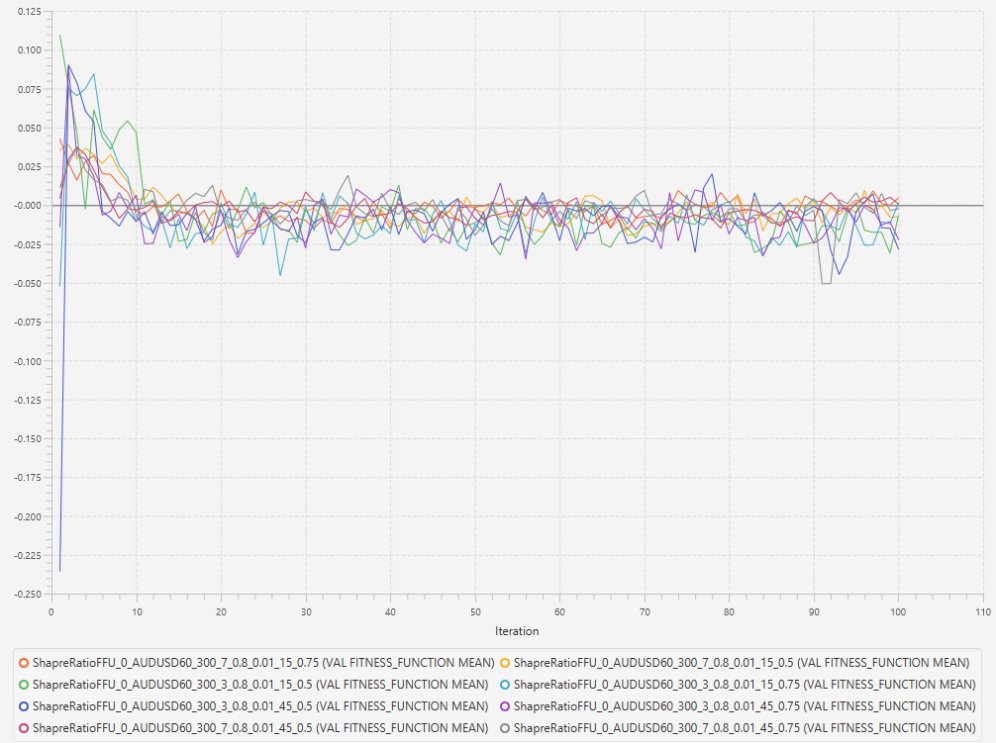


Figure 67: The average maximum Sharpe ratio value at each iteration of the Genetic Algorithm using validation data. Vertical axis is fitness function value

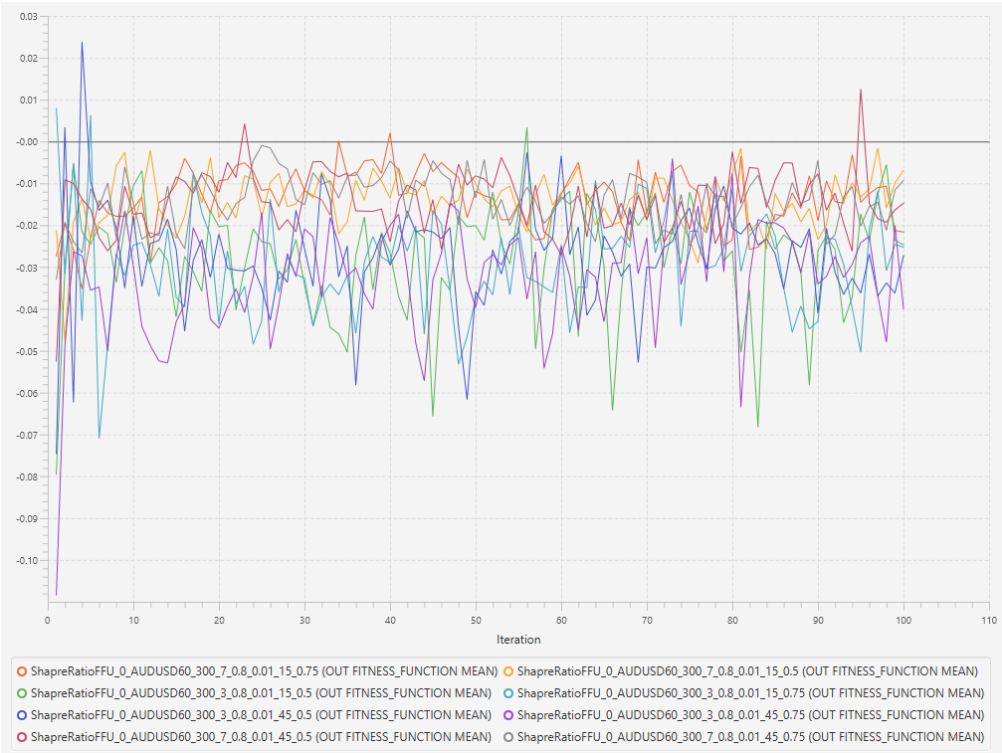


Figure 68: The average maximum Sharpe ratio value at each iteration of the Genetic Algorithm using outsample data. Vertical axis is fitness function value

Figure 69 shows the return of the trading strategy with the highest Sharpe ratio at each iteration of the Genetic Algorithm on validation and outsample data. The return decreased within 15 iterations of the Genetic Algorithm and reflects the results found using the Sharpe ratio. Similar to the Genetic Algorithm using the return performance metric as a fitness function, the Genetic Algorithm did not produce good trading strategies with desirable Sharpe ratio values. Unlike the results reported by Lipinski (2010; 2004; 2010), the use of the Sharpe ratio alone as a fitness function does not improve on the reduction in fitness reported in the previous chapter.

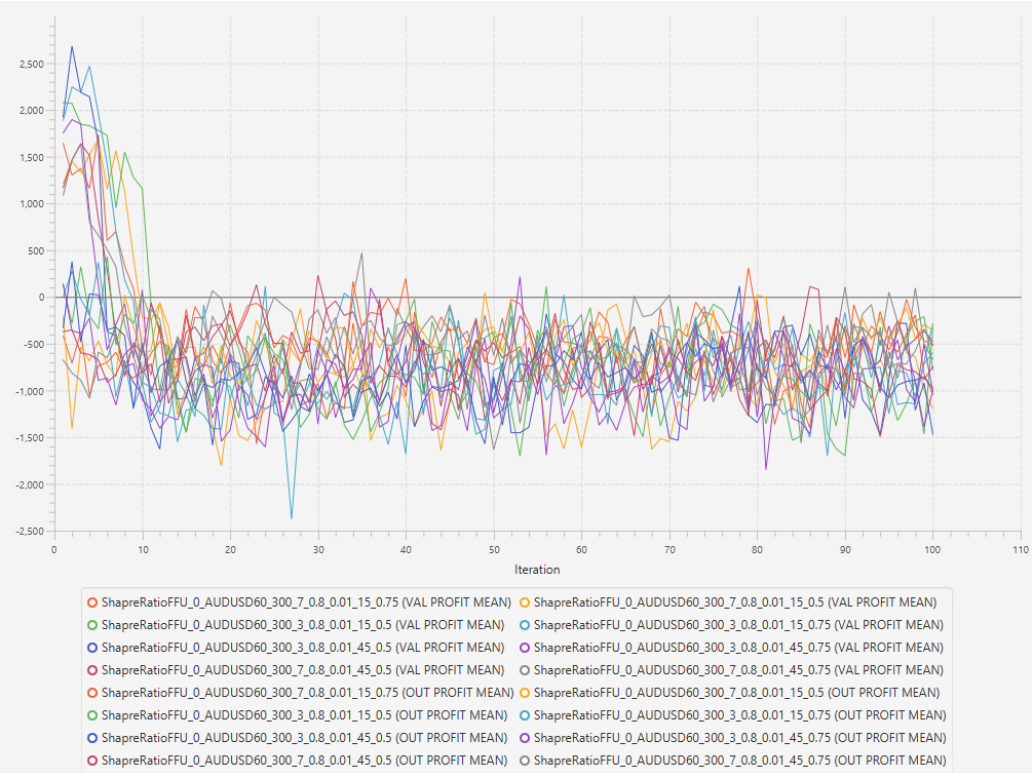


Figure 69: The average maximum return value at each iteration of the Genetic Algorithm using validation and outsample data. Vertical axis is average maximum return

7.2.2 Sortino Ratio Fitness Function

Figure 70 shows the average performance of the fittest trading strategy individuals (those which maximise the Sortino ratio performance metric) at each iteration of the Genetic Algorithm using the insample dataset. Figures 71 and 72 show the fittest trading strategies using validation and outsample data. Similar to the results of the return and Sharpe ratio fitness functions the maximum fitness in the population dropped from the initial population

and the population converged within 15 iterations. The Sortino ratio values obtained from the validation dataset however increased before the population converged below the value of 0.1. Even though the there was an increase in fitness during the initial iterations using on validation data, the Genetic Algorithm did not produce good trading strategies with desirable Sortino values.

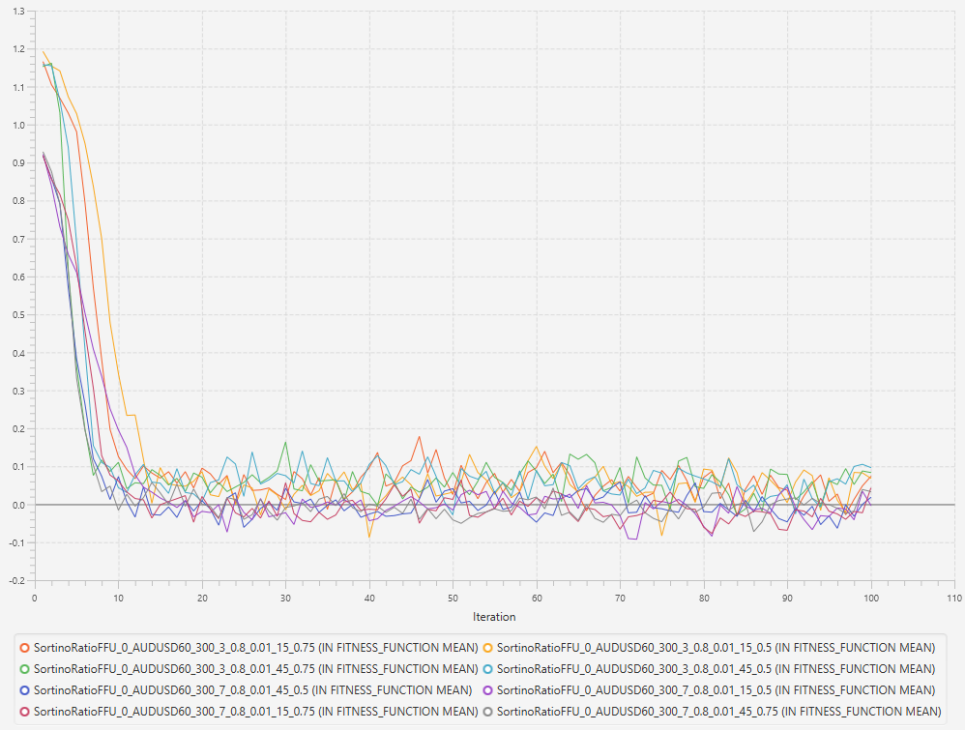


Figure 70: The average maximum Sortino ratio value at each iteration of the Genetic Algorithm using insample data. Vertical axis is fitness function value

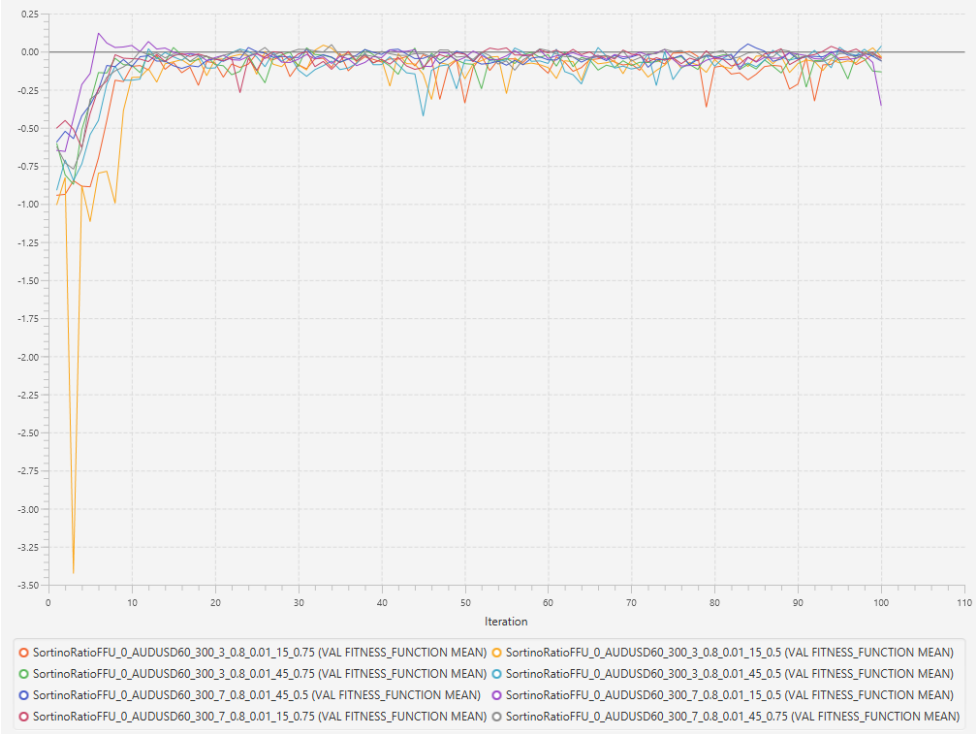


Figure 71: The average maximum Sortino ratio value at each iteration of the Genetic Algorithm using validation data. Vertical axis is fitness function value

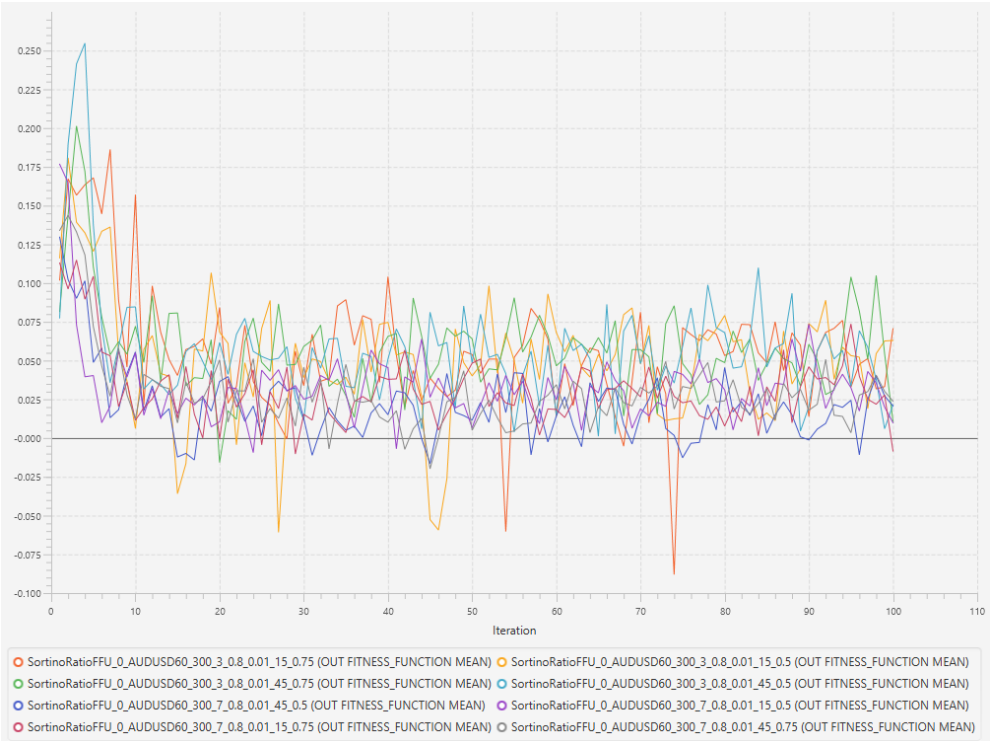


Figure 72: The average maximum Sortino ratio value at each iteration of the Genetic Algorithm using outsample data. Vertical axis is fitness function value

Figure 73 shows the return of the trading strategy with the highest Sortino ratio at each iteration of the Genetic Algorithm on validation and outsample data. The return decreased within 15 iterations of the Genetic Algorithm and reflects the results showing the Sortino ratio.

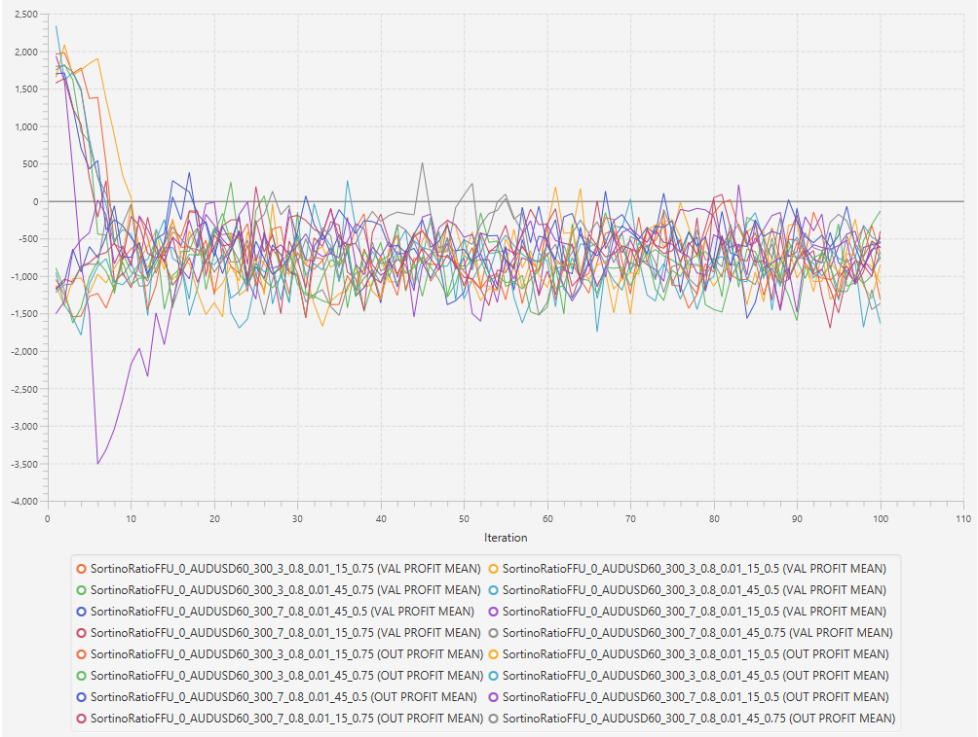


Figure 73: The average maximum return value at each iteration of the Genetic Algorithm using validation and outsample data. Vertical axis is the average maximum return

7.2.3 Trade Success Rate Fitness Function

Figure 74 shows the average performance of the fittest trading strategy individuals, i.e. those which maximise the trade success rate performance metric at each iteration of the Genetic Algorithm using the insample dataset. Figures 75 and 76 show the fittest trading strategies using validation and outsample data. Similar to the disappointing results of the return, Sharpe ratio and Sortino ratio fitness functions, the maximum fitness in the population dropped from the initial population and the population converged within 15 iterations.

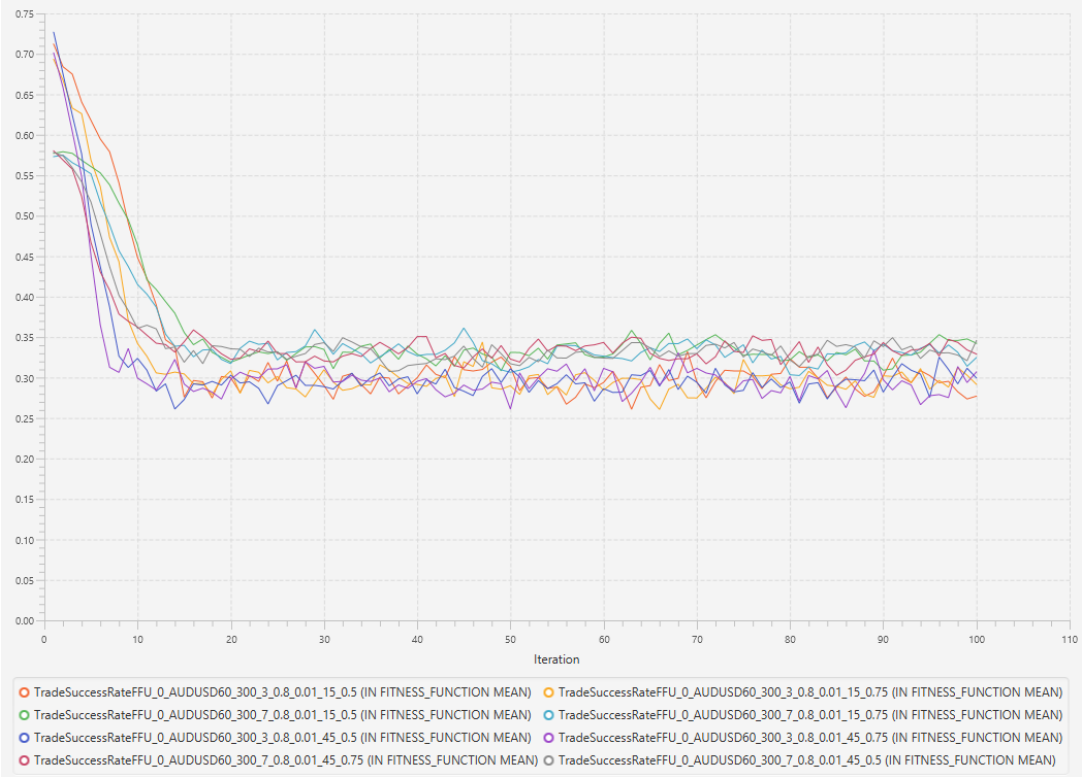


Figure 74: The average maximum trade success rate value at each iteration of the Genetic Algorithm using insample data. Vertical axis is fitness function value

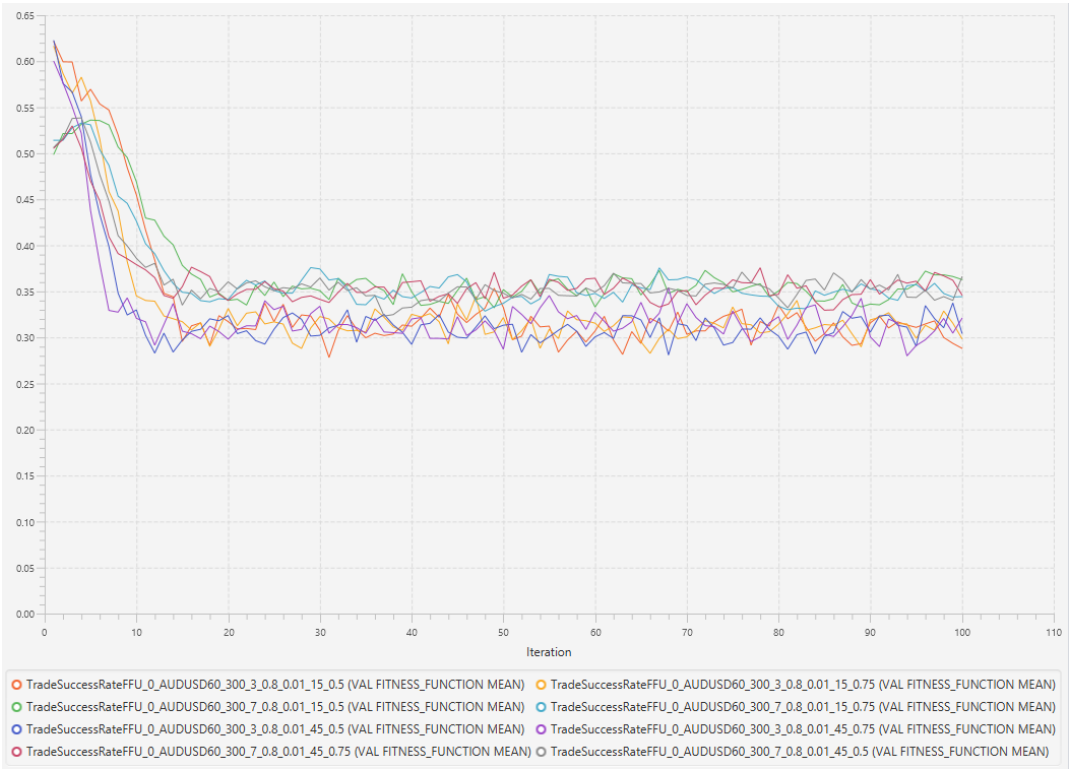


Figure 75: The average maximum trade success rate value at each iteration of the Genetic Algorithm using validation data. Vertical axis is fitness function value

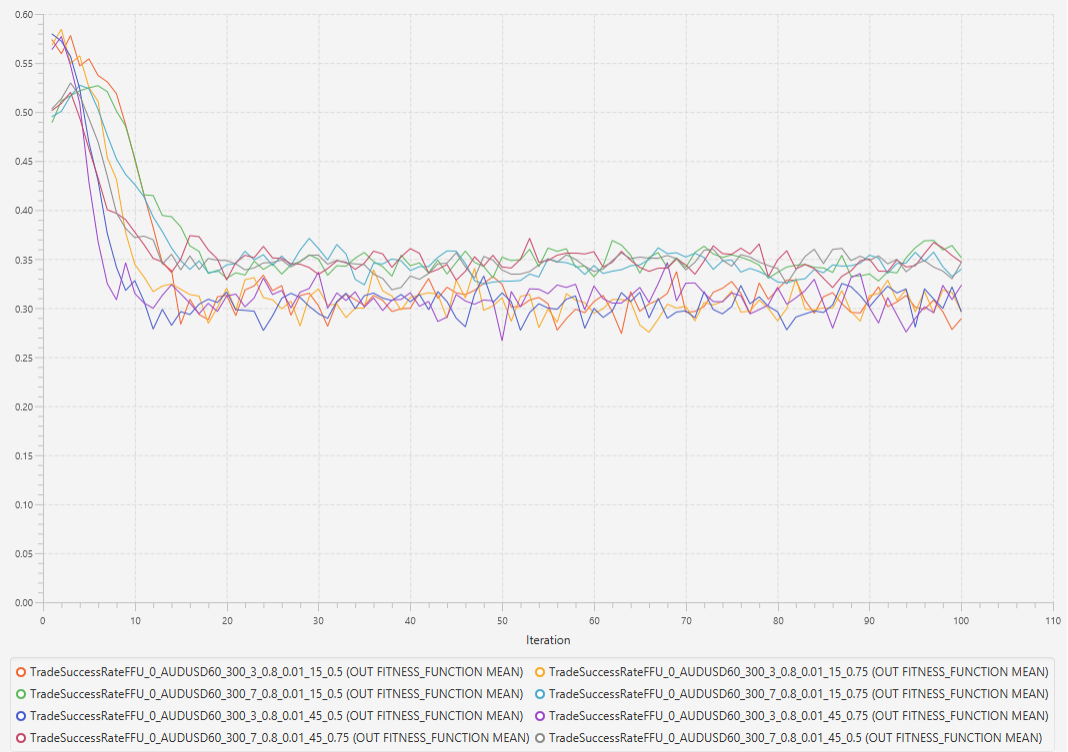


Figure 76: The average maximum trade success rate value at each iteration of the Genetic Algorithm using outsample data. Vertical axis is fitness function value

Figure 77 shows the return of the trading strategy with the highest trade success rate performance metric at each iteration of the Genetic Algorithm on validation and outsample data. The return decreased within 15 iterations of the Genetic Algorithm and reflects the results showing the trade success rate metric.

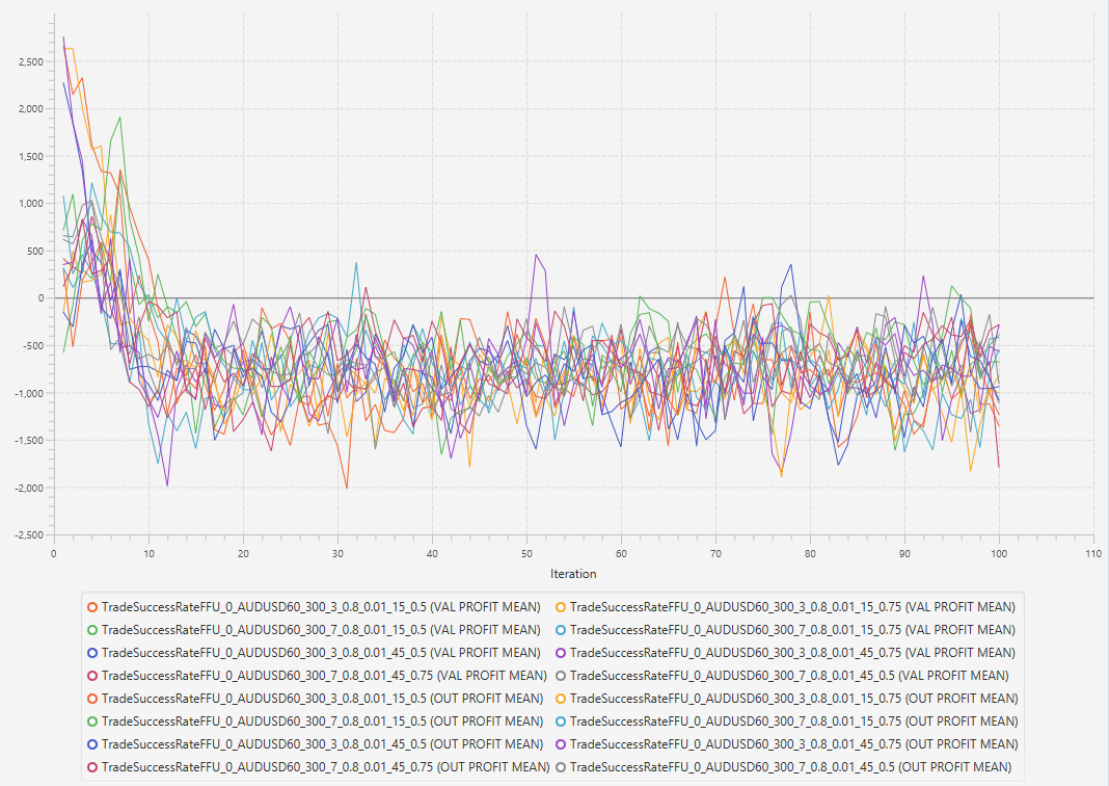


Figure 77: The average maximum return value at each iteration of the Genetic Algorithm using validation and outsample data. Vertical axis is average maximum return

7.2.4 Summary of the Use of Alternative Fitness Functions

The experiments in this section investigated the use of the Sharpe ratio, Sortino ratio and the trade success rate performance metric as a fitness function for the Genetic Algorithm. The previously reported problems with the Genetic Algorithm still persist. The population converges quickly and the average fitness converges at a lower value than possessed by the initial population. The fittest individuals in the population never increase to desirable fitness values on insample data so that it would be worthwhile validating them using validation data and then on outsample.

7.3 Reducing the Size of the Pool of Technical Analysis Interpretations

Suppose that a smaller pool of technical analysis interpretations were to be created. If there is a member of the pool possessing dominant good or bad characteristics, that member would be expected to typically accelerate the Genetic Algorithm towards a good or bad

conclusion. This section explores a smaller pool of only 100 technical analysis interpretations where populations of 100 trading strategies are created. The 8 different parameter setting configurations of the Genetic Algorithm are shown in Table 18. Figure 78 shows the average of 100 Genetic Algorithm runs of the fittest trading strategy on AUDUSD insample, validation and outsample data.

Stage	Parameter	Parameters explored
All	Size of technical analysis interpretation pool	100
All	Population size	100
All	Trading strategy size	3
Selection	Tournament win probability p	0.5 and 0.75
Selection	Tournament size k	5% and 15% of the population size parameter
Selection	Fitness function	Return
Crossover	Crossover probability	0.8
Mutation	Mutation rate	0.01 and 0.1

Table 18: Genetic Algorithm parameter settings

The results are similar to previous experiments, albeit with convergence towards a worse average fitness population being somewhat slower. The population's fitness decreases from the outset. It seems unlikely from these results that the previous poor convergence reported was due to population size, and instead the effectiveness of the fitness function to locate optima within its associated search space seems the more likely cause.

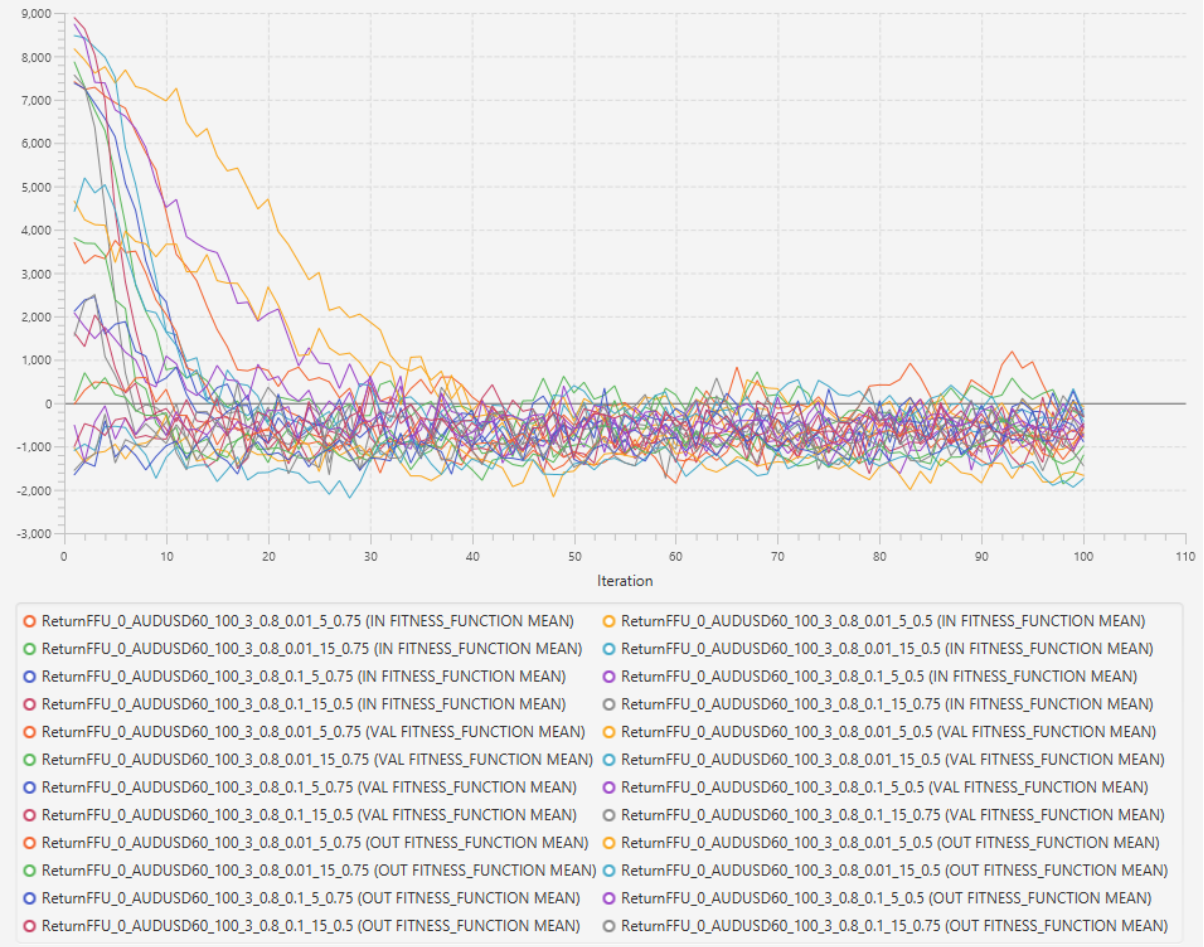


Figure 78: The average maximum return value at each iteration of the Genetic Algorithm using insample, validation and outsample data.

7.4 Changing the Market Dataset

To ensure that the difficulties reported do not arise, or have been contributed to, by some artefact of the AUDUSD dataset, the following experiment applies the Genetic Algorithm using the parameter settings in Table 19 to the EURUSD, GBPUSD, USDCAD, USDCHF and USDJPY foreign exchange market datasets over the same dates as the AUDUSD dataset used in the previous experiments.

Stage	Parameter	Parameters explored
All	Size of technical analysis interpretation pool	10,000
All	Population size	300
All	Trading strategy size	3
Selection	Tournament win probability p	0.5 and 0.75
Selection	Tournament size k	5% and 15% of the population size parameter
Selection	Fitness function	One of Return, Sharpe ratio, Sortino ratio or Trade Success rate
Crossover	Crossover probability	0.8
Mutation	Mutation rate	0.01

Table 19: Genetic Algorithm parameter settings

Figure 79 shows the average performance of the fittest trading strategy individuals which maximise the return performance metric at each iteration of the Genetic Algorithm using the insample dataset. Figures 80 and 81 show the fittest trading strategies using validation and outsample data. Similar to previous results, the fitness of the fittest trading strategies using insample data decreases and the population converges within 15 iterations. The fittest trading strategies used on validation and outsample data did not increase to desirable fitness values and fitness remained close to fitness of the initial population. The same occurs for the Sharpe ratio, Sortino ratio and Trade Success rate fitness functions, and so the results for these are not shown.

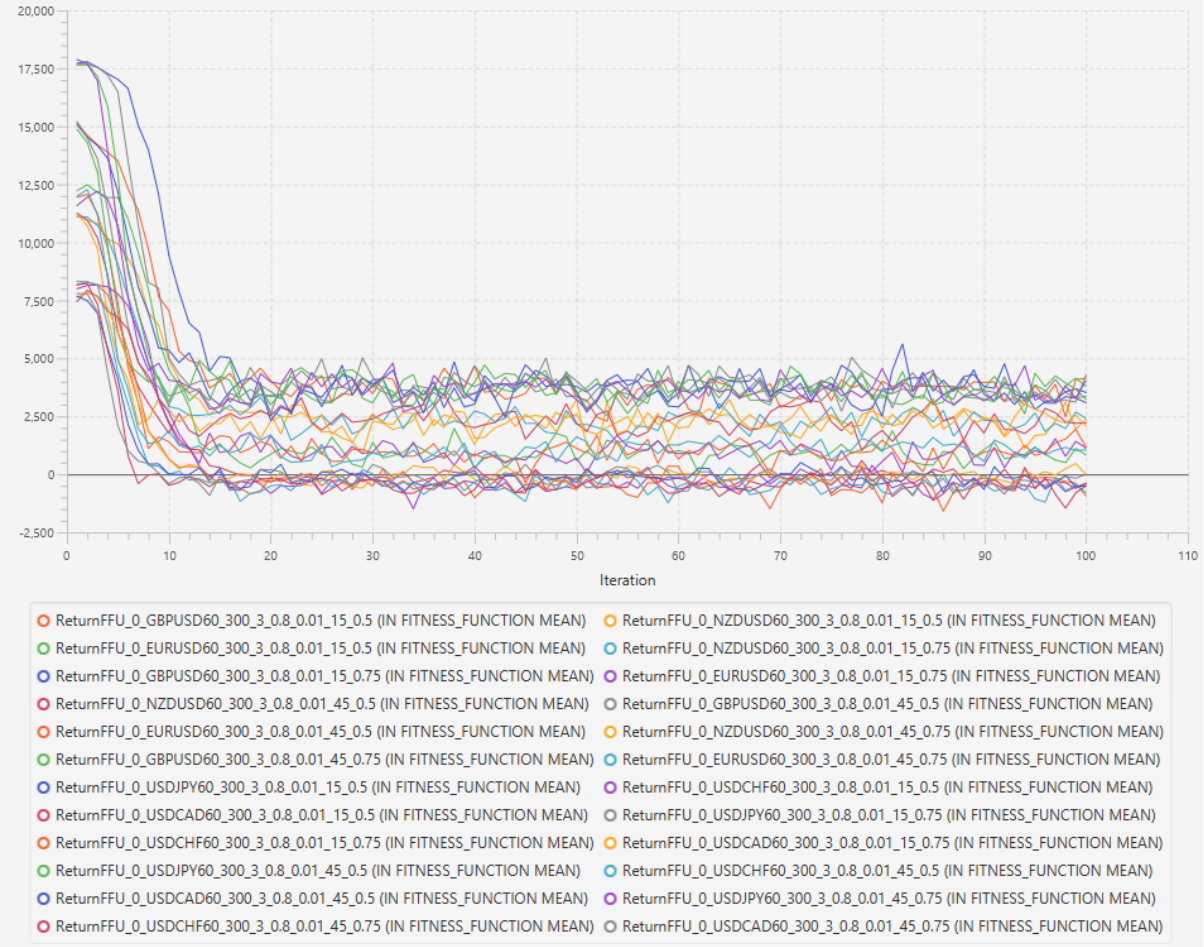


Figure 79: The average maximum return value at each iteration of the Genetic Algorithm using insample data. Vertical axis is fitness function value

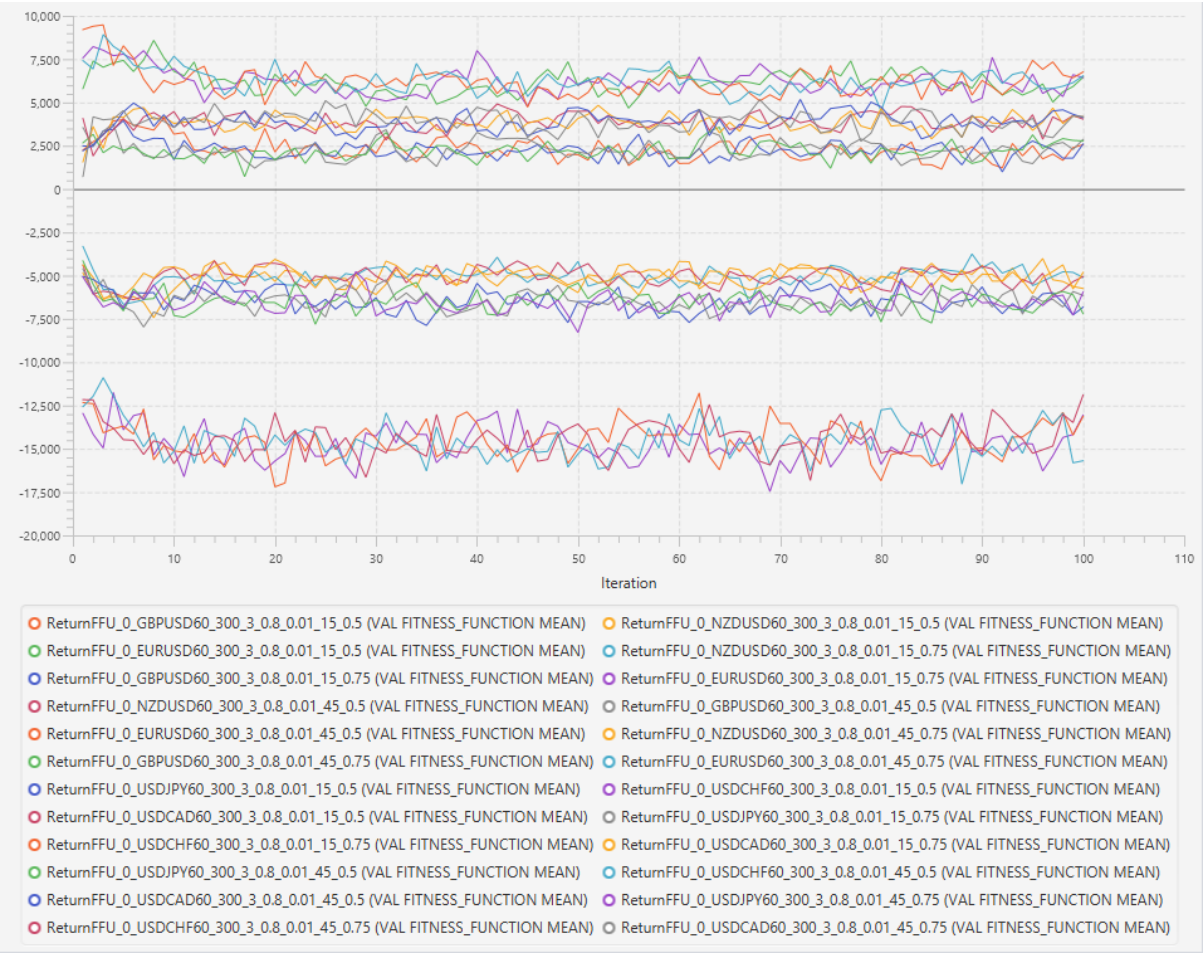


Figure 80: The average return of the fittest trading strategies on insample data at each iteration of the Genetic Algorithm using validation data. Vertical axis is fitness function value

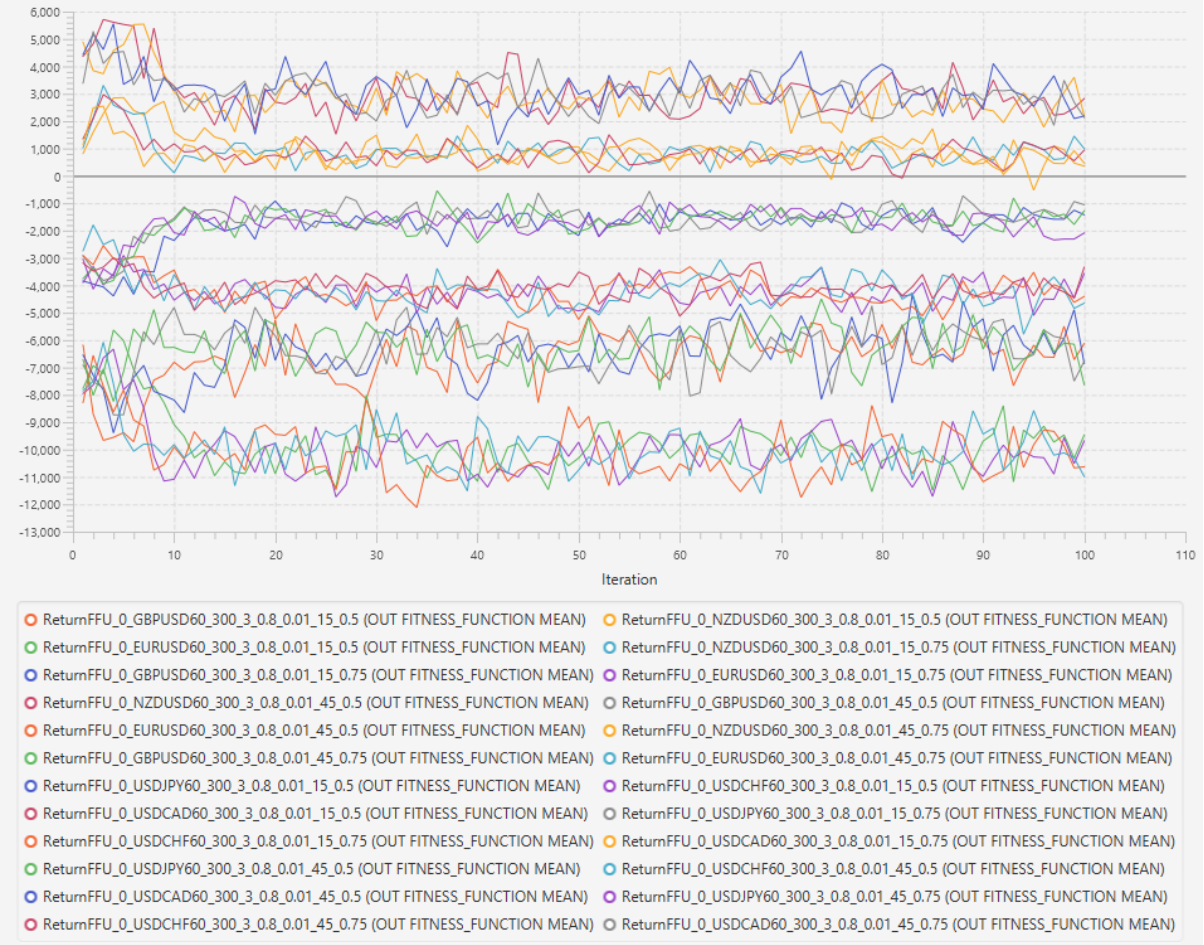


Figure 81: The average return of the fittest trading strategies on insample data at each iteration of the Genetic Algorithm using outsample data. Vertical axis is fitness function value

7.5 Conclusions

The Genetic Algorithm has many configurable parameter settings. This chapter and the previous chapter experimented with many of these settings: different performance metric fitness functions, different numbers of technical analysis interpretations per trading strategy and different market datasets. The Genetic Algorithm in these experiments failed to produce trading strategies with desirable performance metrics on insample data. The purpose of the validation dataset is to locate the optimal Genetic Algorithm iteration for selecting the best trading strategy before the population memorises the insample dataset. The purpose of the outsample dataset is to evaluate the trading strategies performance on unseen data.

The population of trading strategies in the experiments converged quickly (to low values) and no obvious increase in insample fitness was observed to show the Genetic Algorithm

could be used to create fitter trading strategies than the initial population. The goal of this and the previous chapter was to create good trading strategies as substitutes for traders, and the goal of this thesis is to create an early warning system for bad traders. Had trade histories from real traders been made available for this project, performance metrics could indeed have been determined It is also noted here that real traders can deploy various trading strategies which use more complex information beyond the technical analysis algorithms employed in this thesis. Audits of real traders trade histories, however, comprise sensitive information, hence the use here of trading strategies as substitutes for traders.

It seems likely that the fitness functions employed are not suitable for locating optima within their associated search spaces. Financial data are noisy, and some mechanism that would produce a smoother search space landscape would be beneficial. It is plausible that a multi objective fitness function would help guide the search, and might employ a weighted sum of the Return, Sharpe ratio and Sortino ratio and other performance metrics (as described in Chapter 4). Such weighting would reflect a relative importance of the performance metrics.

The literature on optimising trading strategies focuses on tweaking the parameter settings, see Section 2.7.1.7. The current work and the work of Lipinski (2010; 2004; 2010) focuses on finding combinations of technical analysis interpretations to produce good trading systems. Further experimentation could involve changing the representation of the trading strategy model to a binary chromosome representation, as was used by Lipinski. However, the results presented by Lipinski are not comprehensive and focus on specific and narrow ranges of data. It may still be the case that even were encouraging results for insample data to be found, overfitting could occur and be detected in validation. The overriding concern at this stage is the generation of effective multi objective fitness functions, and the approaches taken to modelling strategies here and by Lipinski do not progress towards that goal.

In the next chapter, a more direct approach to the classification of trading strategies will be taken, rather than focusing on the generation of trading strategy pools which can then subsequently be classified. This approach will also generate effective multi-objective fitness functions from trading strategy performance metrics.

Chapter 8 – Classifying Technical Analysis Interpretations using Adaboost

Previous chapters have highlighted the difficulty in analysing noisy data, and the need for the generation of effective multi-objective fitness functions informed by trading strategy performance metrics. This chapter outlines the system and implementations used for classifying technical analysis interpretations based on the Adaboost algorithm. Analysis is given here of the effectiveness of Adaboost classifiers for classifying and identifying profitable and unprofitable technical analysis interpretations, and thereby assesses the effectiveness in classifying simple trading strategies, a proxy for human traders.

This chapter will first outline the classification systems used experimentally in this project. Then the data the systems will be applied to will be described. Special attention is paid to the problem of overfitting. This chapter will then examine each of the binary classification performance values (described in Section 2.5) found as the Adaboost algorithm iterates, and will report the receiver operating characteristic as described in Section 2.6.

8.1 Adaboost Classification

Adaboost, unlike other classification methods such as artificial neural networks (Section 2.4.1) and support vector machines (Section 2.4.2), is a *white box* approach to classification. That is, the mechanisms used are explicit, and decisions can be explained. Decisions made by an Adaboost classifier are therefore easier to understand and can be interpreted by considering the total weight assigned to each variable, this will be explored in 11.7. This is distinct from a *black box* approach to classification where it is difficult to explain or directly interpret a classifier's decisions (Murphy, 2012). Indeed, it is difficult to see how strategies to improve trader performance can be found otherwise. Adaboost will enable direct scrutiny of the weighting of metrics employed. The industrial partner has experienced mixed success with black-box approaches applied to financial market forecasting and therefore has reason to doubt the trading decisions of a black box method.

8.1.1 Adaboost in the Context of Technical Analysis Interpretations

The Adaboost algorithm (outlined in Section 2.4.3.2) is an ensemble method that attempts to combine many weak classifiers to create a more predictive classifier called a *strong*

classifier. Adaptive boosting (Adaboost for short) adaptively corrects the classification errors of the current classifier at each iteration of the algorithm. To achieve this, data points in the training set are assigned weights that indicate how important that data point is in being classified. These weights are adjusted at each iteration of the algorithm. Adaboost is used to construct a strong classifier of the form,

$$H = sign\left(\sum_{i=0}^{n} \propto_i h_i\right)$$

where α_i is the weight of the ith weak classifier, h_i is the value of the ith weak classifier, H is the strong classifier value, and 'sign' is the sign function which indicates whether the summation is positive or negative,

$$sign(x) = \begin{cases} -1 \text{ if } x < 0 \\ 0 \text{ if } x = 0. \\ 1 \text{ if } x > 0 \end{cases}$$

An individual performance metric with a threshold is considered here to be a weak classifier; the threshold, a particular value which is a floor or ceiling to values requiring classification, determines whether the weak classifier outputs the value -1 or 1. The weak classifier classifies technical analysis interpretations as good or bad based on whether the interpretation's performance metric satisfies the threshold. The final strong classifier is a weighted sum of each weak classifier. This translates to a classifier that classifies technical analysis interpretations based upon multiple performance metrics, where each performance metric has a different impact on the final classification decision; it is intended to be a more effective classifier than any of the individual weak classifiers.

The training dataset consists of performance metrics derived from one segment of historical data and a label metric derived from the next segment of data within the historical dataset. The performance metrics are outlined in Chapter 4 and are calculated from 10,000 different technical analysis interpretations. Each technical analysis interpretation has a random parameter setting.

Adaboost classifiers have clear decision-making advantages over human experts. Adaboost can decipher patterns within a training set with many *variables*, i.e. with a substantial number of performance metrics. The Adaboost algorithm is also able to cope with a large

data set, i.e. a large number of technical analysis interpretations in the training set. Although human experts, on the other hand, can utilise higher-level information such as fundamental analysis and charting, they cannot do so systematically, nor can they always justify or analyse such essentially intuitive processes.

8.1.2 Market Data and Validation

The results in this chapter are derived from hourly AUDUSD, EURUSD, GBPUSD, USDCAD, USDCHF and USDJPY foreign exchange data from 19th of May 2014 to the 23rd of January 2016. Each market dataset contains 10,344 data points and are the same market datasets that were outlined in Section 6.2.

Each of the foreign exchange markets are split into 4 segments as outlined in Table 20. The segments are then used to create a training dataset, a validation dataset and a outsample dataset to create and test the effectiveness of Adaboost classifiers.

Segment	Date from	Data to	Number of data points
1	19th of May 2014	15th of October 2014	2586
2	15th of October 2014	20th of March 2015	2586
3	20th of March 2015	20th of August 2015	2586
4	20th of August 2015	23rd of January 2016	2586

Table 20: Information about each market data segment

The training dataset is made up of performance metrics of technical analysis interpretations, which are derived from the first segment of market data, and a corresponding performance metric label, which is obtained from the second segment. Having created a classifier using the training dataset, the classifier can then be used to classify technical analysis interpretations using performance metrics derived from the second segment in Table 20 onwards. In this chapter the classifiers are used on validation data and outsample data. The segments used for the training set, validation set and outsample set are outlined in Table 21.

	Derive all metrics from	Classify
Training dataset	Segment 1	Segment 2
Validation dataset	Segment 2	Segment 3
Outsample dataset	Segment 3	Segment 4

Table 21: Segments used for each dataset

Deploying classifiers that have overfitted the training dataset is a concern and occurs when the classifier has memorised the training dataset. The purpose of the validation dataset is to detect optimal classifiers by choosing for low bias and low variance (see Section 2.3). The outsample dataset is used to test the classification performance of the chosen classifier.

Combining all the technical analysis interpretations of each market makes a total of 70,000 technical analysis interpretations. When these 70,000 interpretations were used on the segment 1, 41% were found to be profitable, 49% were not profitable and 10% were break even. For the segment 2, 26% were profitable, 83% were unprofitable and 1% was neither profitable nor unprofitable. For segment 3, 35% were profitable, 64% were unprofitable and 1% was neither profitable nor unprofitable. The percentage of profitable and unprofitable technical analysis interpretations depends on the market segment used.

8.1.3 Performance Metrics

As mentioned in Chapter 4, financial traders backtest their trading strategies and judge their performance based on performance metrics such as profit factor, drawdown and number of trades. In this chapter, the Adaboost classifiers will classify whether a trading strategy is 'good' or 'bad' using all of the performance metrics outlined in Section 4.8 for each trading strategy.

In this chapter, technical analysis interpretations are classified here as potentially good or bad based on whether or not they are profitable. To do this, the Adaboost algorithm attempts to separate the training dataset using the following categorization,

$$Classification = \begin{cases} Good & if\ Return > 0\ pips \\ Bad & if\ Return \leq 0\ pips \end{cases}$$

The next section will outline the techniques used to reduce overfitting before performing the experiments in Section 8.3.

8.2 Reducing Overfitting

Bootstrap aggregation, as described in 2.4.3.1, is a technique used to reduce overfitting and which assists in the production of a more generalized classifier. The technique combines the classifications of several classifiers and produces a final classification that is the most frequent. Each individual classifier is trained on a random subsample of data points with the possibility of picking the same data point more than once.

Feature bagging, as described in 2.4.3.1, is another ensemble method which helps to reduce overfitting of high variance models with low bias. It is possible that individual classifiers in the ensemble are too similar thus producing almost identical classifications. The random subspace method helps to solve this problem by training each classifier in the ensemble on a random subset of features (performance metrics) at each iteration of the AdaBoost algorithm.

It would be expected that each of the individual classifiers would overfit in different ways, as each individual classifier is trained using a different subsample of data points and/or on a different subset of features from the training dataset. Each of the individual classifiers should contain parts of the general classification and it is expected that a general classification emerges from the combination of individual classifiers.

Both of these techniques will be used in the experimentation sections of this chapter.

8.2.1 Illustration of AdaBoost with Bootstrap Aggregation

A Java application was created to validate the Adaboost code and visualise its classification using the bootstrap aggregation technique to reduce overfitting. Figure 82 shows an artificially created training dataset of 10,000 data points. The data points in the artificial dataset can be categorized into two classifications, red and blue. The classification boundary is noisy as there is overlap between the red and blue data points. The data points depend on the two variables that mark the *x* (horizontal) and *y* (vertical) axes of the image. A *decision stump* is a weak classifier that thresholds a variable (*x* or *y*) above or below a particular value.

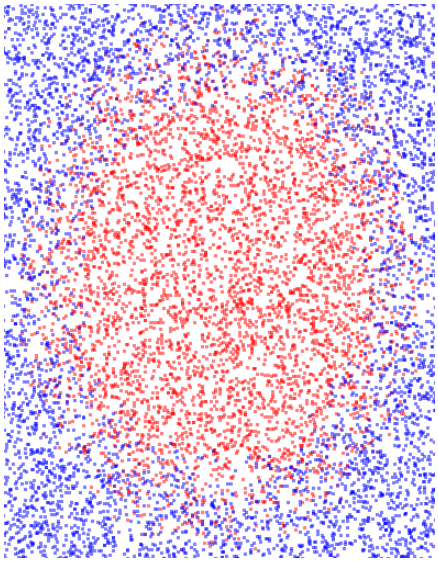


Figure 82: Dataset – 10,000 data points

Individual classifiers created by the Adaboost algorithm using a subsample of 1,000 data points from the training dataset are shown in Figure 83. The majority of the blue and red regions are classified correctly, but this classification is unreliable as it varies depending on the subset of the training dataset used by the Adaboost algorithm. The general classification of the overlap of red and blue points is not achieved.

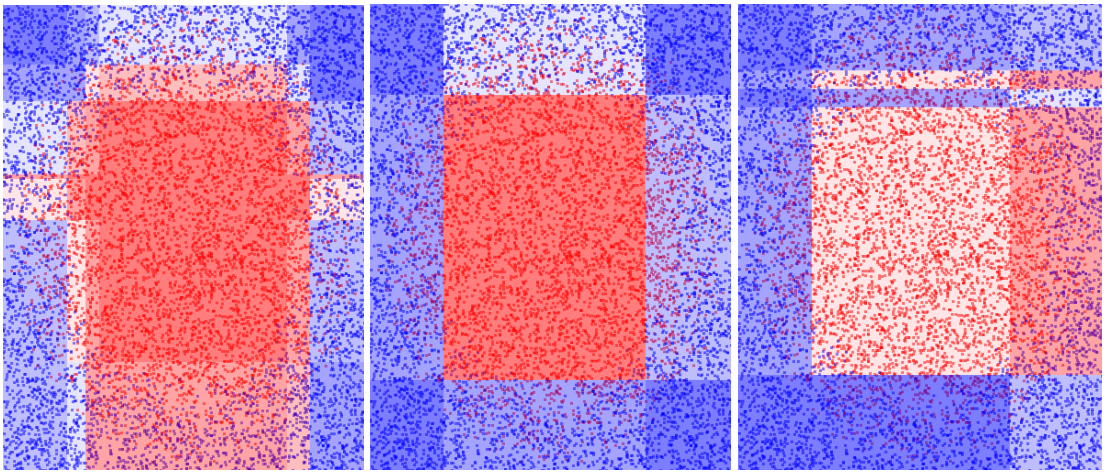


Figure 83: 3 individual classifiers each with a subsample of the dataset of 1,000 data points

By combining 100 individual classifiers in a majority vote decision, the resultant classifier is able to correctly separate the blue and red data points (see Figure 84). It is important to note that each classifier is created from a subsample of 1,000 data points from the initial

pool of 10,000 data points otherwise they would produce the same classifier. The blue and red classification regions of the image have varying transparencies to reflect the number of classifiers that have classified the area as a specific classification (red or blue). The classification boundary appears almost white because the number of votes for both the blue and red classification are about equal.

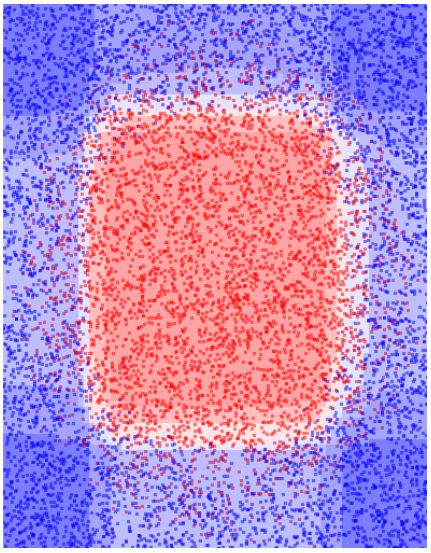


Figure 84: 100 individual classifiers combined in majority vote

Visual inspection shows that the whiter regions lie close to the noisy overlap between the blue and red data points. The darker red or blue classification regions of the image contain the most votes for a particular classification. This implies that the more votes a data point has for a particular classification, the more probable it is that the data point belongs to that classification in this example.

The following experiments in this chapter will use the random subspace method and bootstrap aggregation technique to reduce the chance of overfitting.

8.3 Test and Results

This section explores the classification of technical analysis interpretations using different Adaboost-based classification systems. Experiments for different individual Adaboost classifiers are analysed and reported, and also for two classification systems in which Adaboost is augmented by mechanisms designed to reduce overfitting (described in Section 432.4.3.1). The first classification system, called the *bootstrap classifier*, uses the bootstrap aggregation technique to help reduce overfitting. The second classification, called the

bootstrap and feature-bagging classifier uses both the bootstrap aggregation and feature bagging technique to help reduce overfitting.

The bootstrap aggregation technique creates a classifier consisting of 51 individual Adaboost classifiers. Each of these 51 classifiers is created using a random subsample of 25% of the technical analysis interpretations from the original training set. Classifiers are then combined in majority vote to produce the final classification.

The feature bagging technique also creates a single classifier consisting of 51 individual Adaboost classifiers. Each of these 51 classifiers is created using a random subsample of 50% of the performance metrics (features) from the original training set. The classifiers are then combined in majority vote to produce the final classification.

The purpose of the experiments in this section is to:

- investigate ways of choosing an optimal Adaboost iteration for selecting the best classification system using the validation dataset;
- find the best performing Adaboost-based classification system for classifying technical analysis interpretations, and in particular to classify unprofitable technical analysis interpretations;
- examine appropriate techniques for reducing overfitting.

8.3.1 Binary Classifier Performance Values

The results of experiments described here will be analysed in terms of the binary classifier performance values introduced in Section 2.5:

- Accuracy and Receiver Operating Characteristic (ROC);
- Precision and recall;
- Negative predictive value and specificity;
- F1 score.

Taking each of the above binary classifier performance values in turn, first it will be established whether, in the training stage, the Adaboost classifier succeeds in learning the

data according to the performance values. Then the results will be analysed for the classification of validation and outsample data, considering each time the individual classifier, the bootstrap aggregation enhancement, and finally the combination of bootstrap aggregation with feature bagging. Each graph in this chapter reports seven separate results derived from each of the foreign exchange markets.

8.3.2 Classifier Accuracy and ROC Performance

This section will report the ROC performance and accuracy of classification systems at each iteration of the Adaboost algorithm during training. Then report validation and outsample performance for the individual classifier, the bootstrap aggregation classification system, and bootstrap aggregation with feature bagging classification system.

8.3.2.1 Accuracy and ROC Performance on Training Dataset

The training dataset is used to create the classifier. The results are displayed in two types of graphs: the first shows the accuracy of the classifier under consideration at each iteration of the Adaboost algorithm using the training dataset; the second shows the receiver operating characteristic (ROC, defined in Section 2.6) results obtained by using the classifiers at the 200^{th} iteration.

Figure 85 shows the accuracy results of individual Adaboost classifiers and, as expected, the accuracy generally increases with iteration. The ROC curves in Figure 86 are visibly far away from the random classifier line, and the areas under the curve (AUC) of the foreign exchange markets AUDUSD, EURUSD, GBPUSD, NZDUSD, USDCAD, USDCHF and USDJPY are 0.83, 0.86, 0.89, 0.87, 0.83, 0.94 and 0.84 respectively. If the individual Adaboost classifiers were random then the AUC would be 0.5 and if the AUC values were below 0.5 the classifiers would be performing worse than random. The average AUC however is 0.87 and all classifiers report an AUC of 0.83 or above.

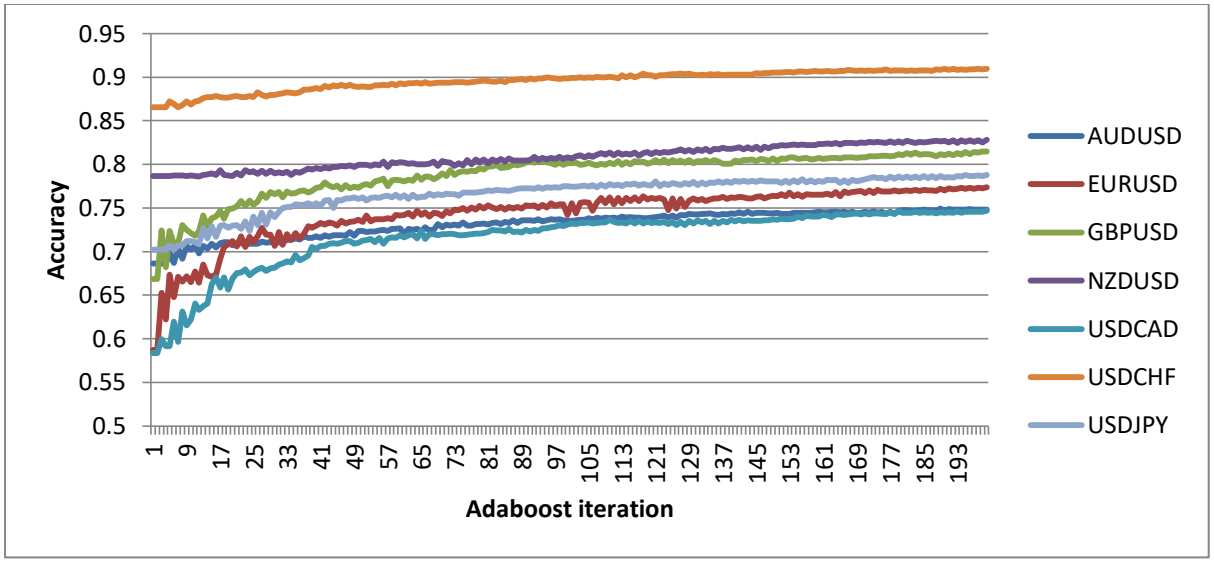


Figure 85: Accuracy of individual classifiers during training

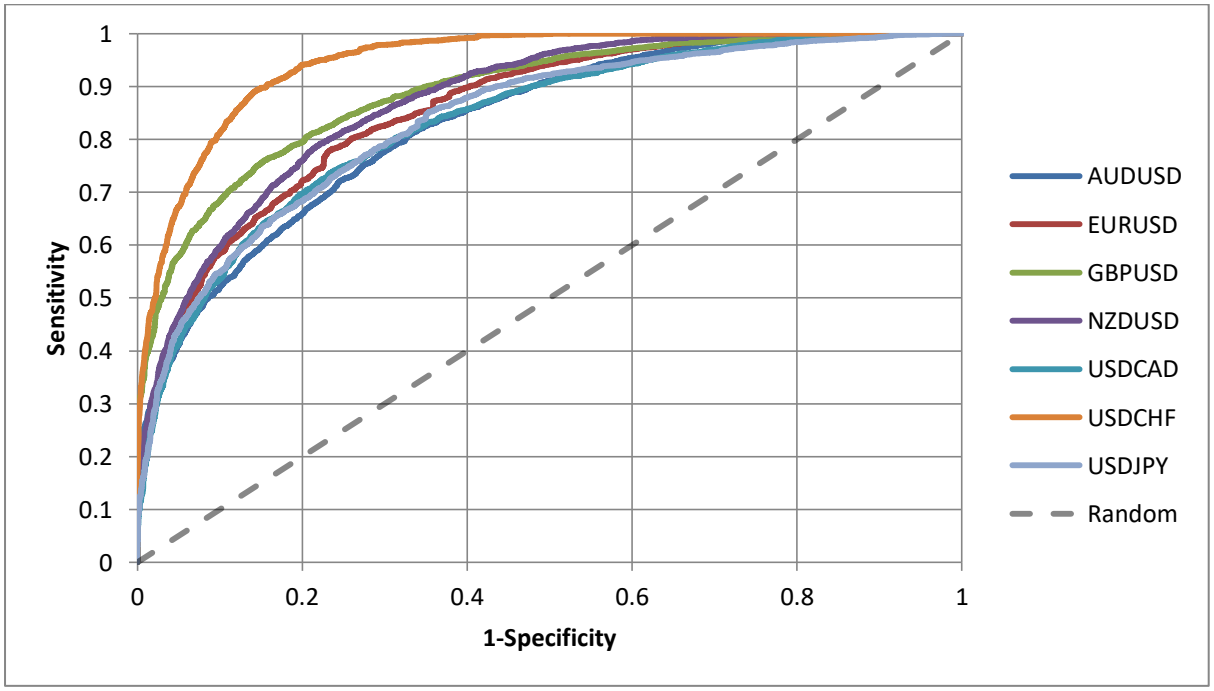


Figure 86: ROC curve of individual classifiers using the training dataset

Figure 87 shows the accuracy results of the bootstrap classifiers. Again, as expected, the accuracy generally increases with each iteration of the Adaboost. These results are smoother than the individual classifier accuracy results which is expected as the classifications of 51 individual Adaboost classifiers are combined. Figure 88 shows the ROC results obtained by using the bootstrap classifiers at the 200th iteration of the Adaboost algorithm. The ROC curves are far away from the random classifier line and the areas under the curve (AUC) of the foreign exchange markets AUDUSD, EURUSD, GBPUSD, NZDUSD,

USDCAD, USDCHF and USDJPY are 0.83, 0.85, 0.88, 0.85, 0.84, 0.91 and 0.83 respectively. The average AUC is 0.86. This shows that the bootstrap classifiers were able to learn the training dataset.

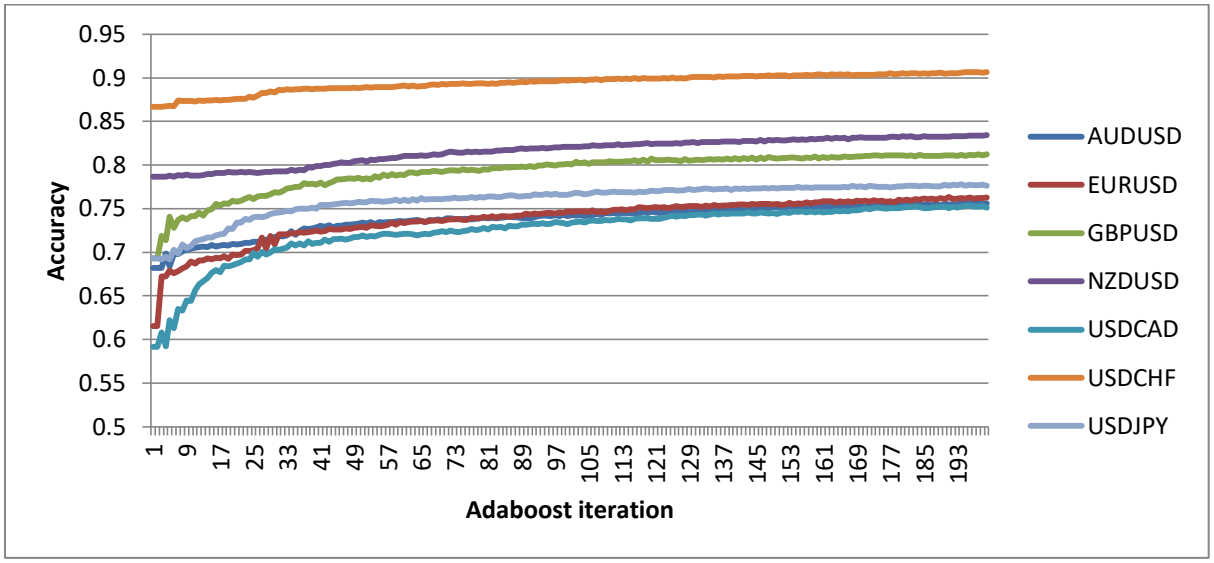


Figure 87: Accuracy of classifiers using the bootstrap aggregation technique during training

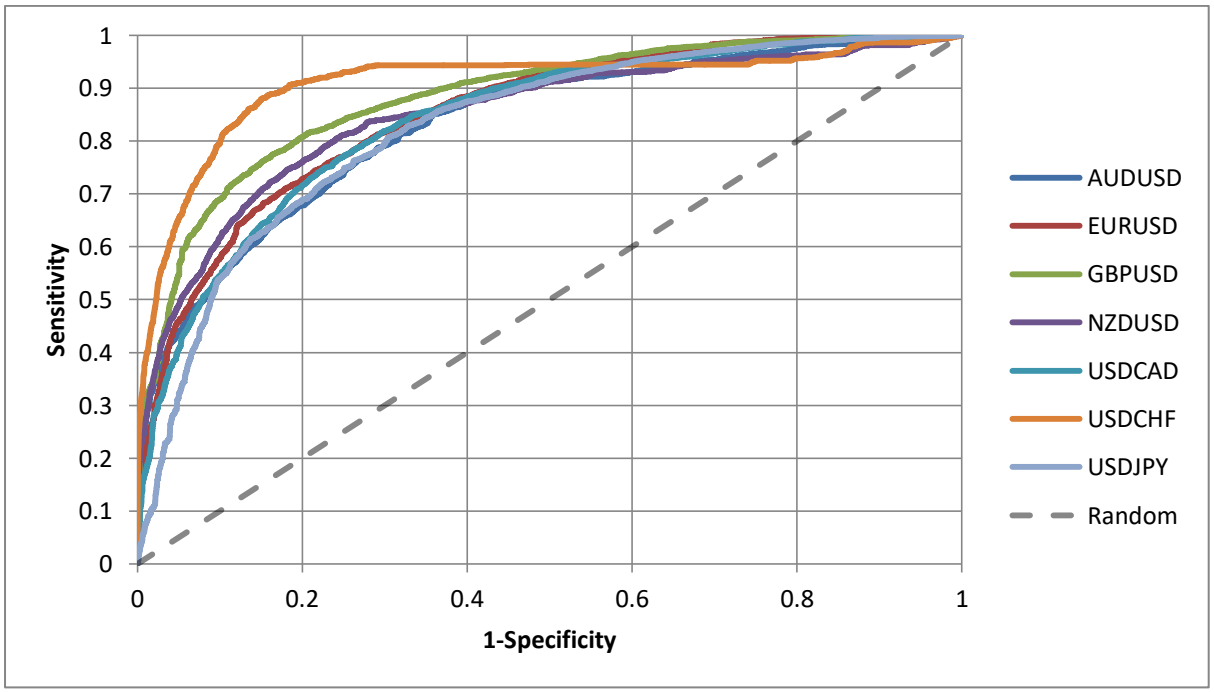


Figure 88: ROC curve of bootstrap classifiers using the training dataset

Figures 89 and 90 show similar results for the accuracy results of the bootstrap and feature-bagging classifiers. The latter shows the ROC results obtained at the 200th iteration of the

Adaboost algorithm. The AUC values are 0.82, 0.84, 0.88, 0.83, 0.83, 0.89 and 0.81 for the respective markets, with an average of 0.84.

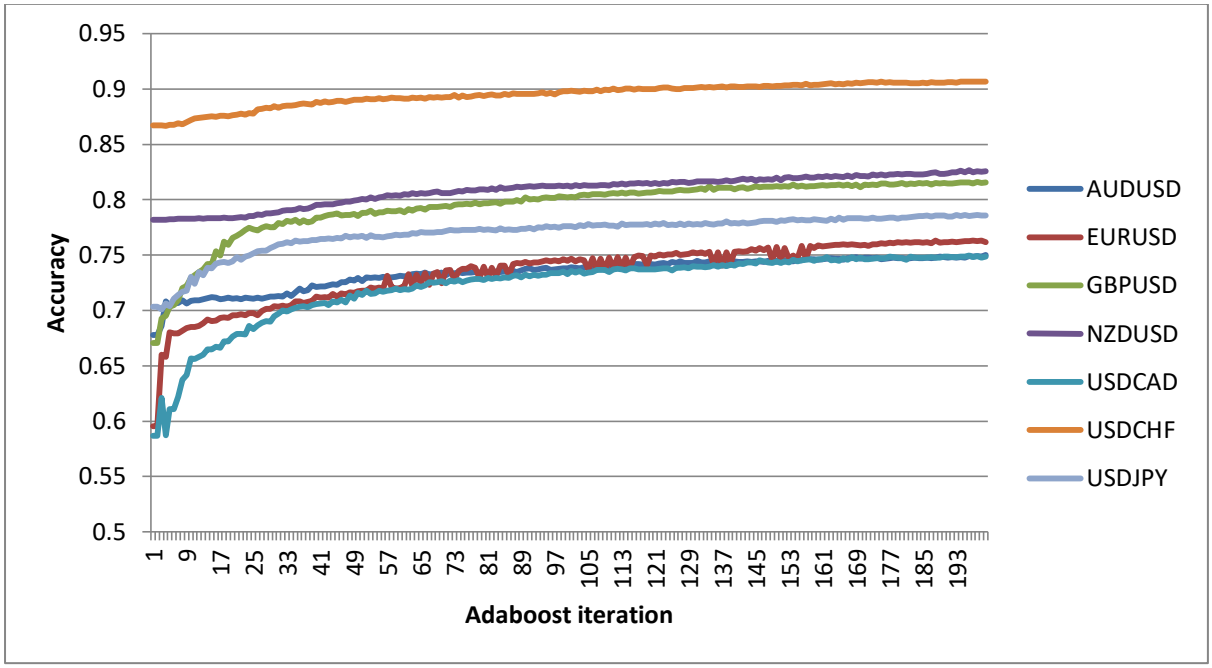


Figure 89: Accuracy of classifiers using the bootstrap aggregation & feature-bagging technique during training

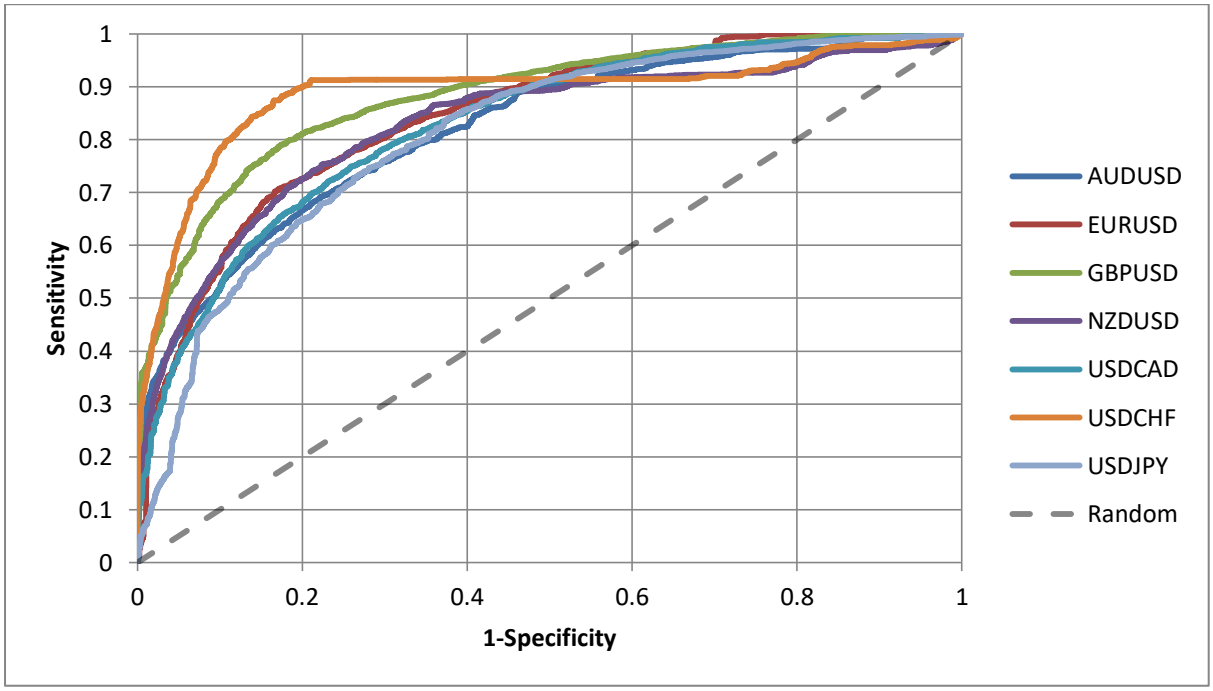


Figure 90: ROC curve of bootstrap and feature-bagging classifiers using the training dataset

The differences between each AUC value and average AUC of each classification system is marginal, e.g. indicating a similar performance in accuracy across all markets. According to the accuracy performance measure, the three Adaboost classifiers have successfully learnt the training dataset.

8.3.2.2 Individual Classifier

Figure 91 shows the accuracy of individual classifiers at each iteration of the Adaboost algorithm using the validation dataset. The individual classifiers classified 31,813 technical analysis interpretations as profitable and 37,303 as unprofitable. The accuracy of individual classifiers using the USDJPY and GBPUSD data increases as the Adaboost algorithm iterates. However, accuracy decreases in the classifiers using AUDUSD, NZDUSD, USDCAD and USDCHF data. Accuracy increases then slightly decreases in the classifier using EURUSD data, which suggests that the optimal classifier is to be found around the 60th iteration. The accuracy of classifiers using AUDUSD and USDCAD data experiences a reduction of about 0.19. This drop suggests that overfitting has occurred in these classifiers. This is likely due to the fact that the Adaboost algorithm is forced into learning heavily weighted data points. To reduce overfitting, optimal classifiers are chosen at the 137th iteration for the classifier using AUDUSD data and the 15th iteration for the classifier using USDCAD data.

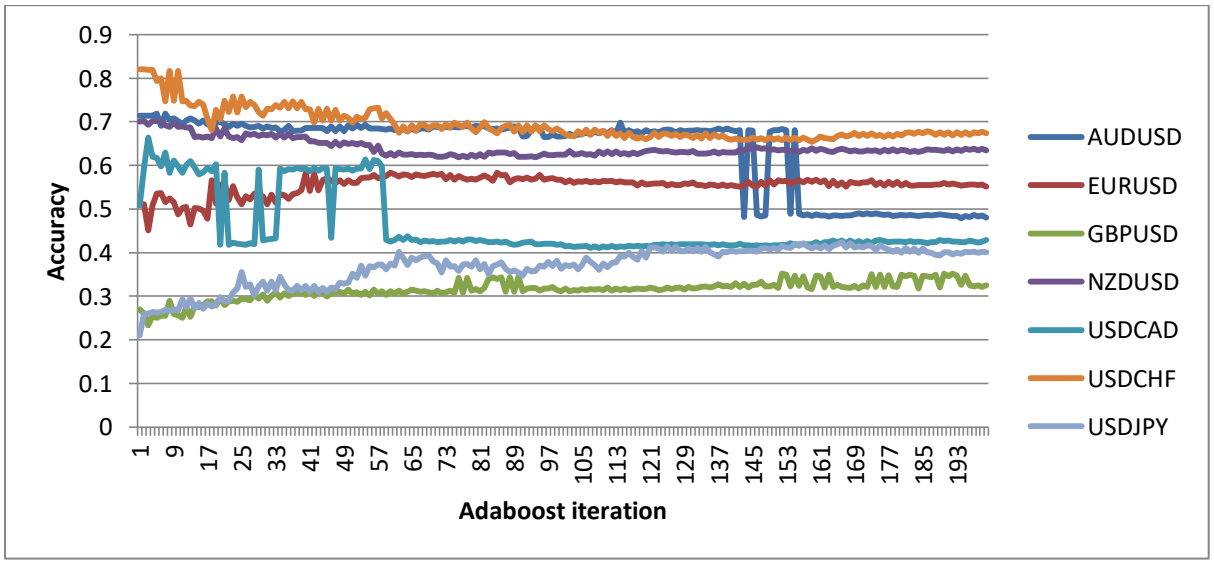


Figure 91: Accuracy of constructing individual classifiers on validation data

Figure 92 shows the accuracy of individual classifiers at each iteration of the Adaboost algorithm using the outsample dataset. The individual classifiers classified 22,029 technical

analysis interpretations as profitable and 46,920 as unprofitable. There are no sharp accuracy drops such as were present in the validation dataset, which suggests that picking optimal classifiers before accuracy drops in the validation dataset is not beneficial. After the 50th iteration, the accuracy values seem to converge, i.e. becoming consistent and fluctuating only slightly in value, given 200 iterations of the Adaboost algorithm. Additionally, 5 out of the 7 classifiers achieved better than random performance at the final iteration of the Adaboost algorithm.

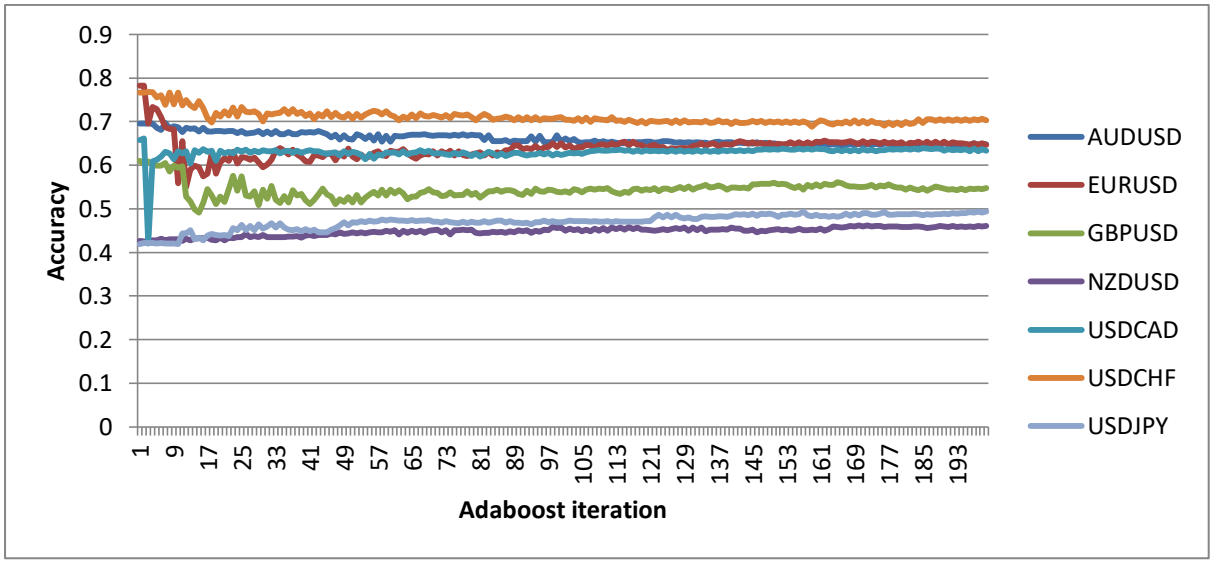


Figure 92: Accuracy of constructing classifiers on outsample data

Figure 93 shows the average accuracy of classifiers using the validation and outsample datasets at each iteration of the Adaboost algorithm. Average accuracy values of classifiers using the outsample dataset are fairly consistent. Due to the overfitting seen in Figure 91, the average accuracy of classifiers exhibits two sharp drops (corresponding to the drops in classifiers for two of the markets).

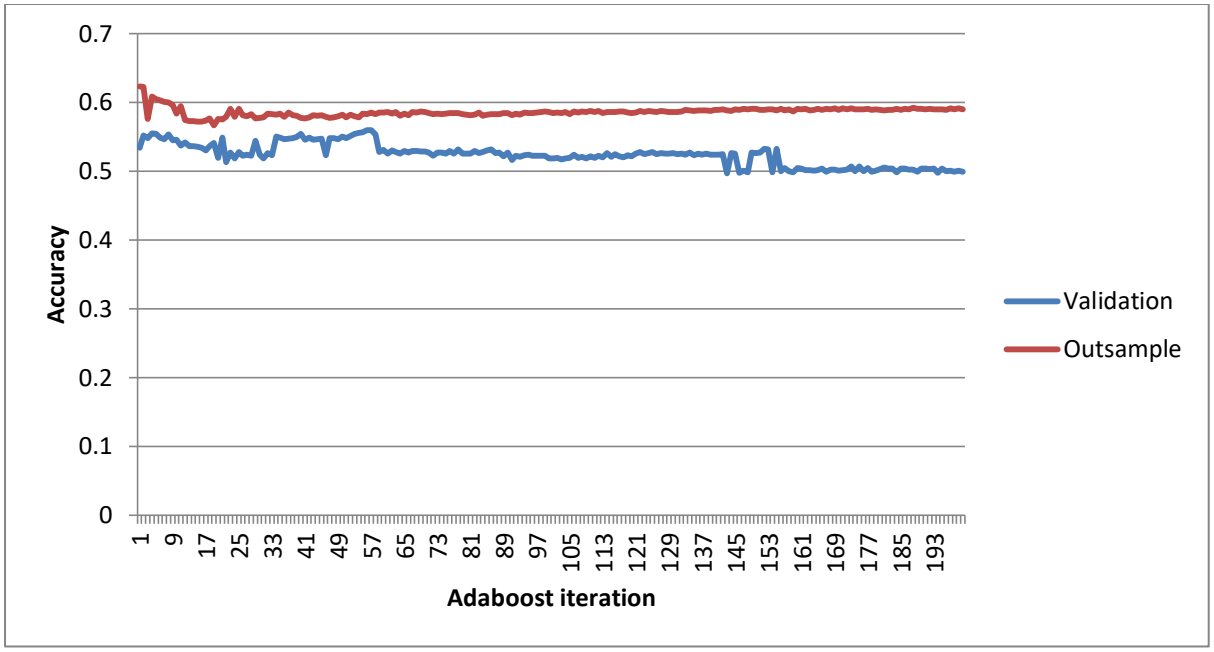


Figure 93: Average accuracy of constructing classifiers on validation and outsample data

The receiver operating characteristic (ROC) curve of individual classifiers on validation data and outsample data can be seen in Figures 94 and 95. The classifiers used on validation data separate technical analysis interpretations better than random, but only just, for the NZDUSD, EURUSD and USDCHF data and perform randomly or worse on AUDUSD, GBPUSD, USDCAD and USDJPY data. However, classifiers used on outsample marginally outperformed these by separating the technical analysis interpretations at best slightly better than random and at worst randomly. The area under the ROC curve (AUC) values for individual classifiers using the training, validation and outsample datasets are summarized in Table 22. The average AUC value of classifiers using the training dataset is 0.87 which indicates that the classifier has separated the technical analysis interpretations, according to the training set, effectively. However, when the classifiers are used on the validation and outsample dataset the AUC values are 0.5 and 0.567 respectively, and so the classifiers perform randomly or slightly better than randomly.

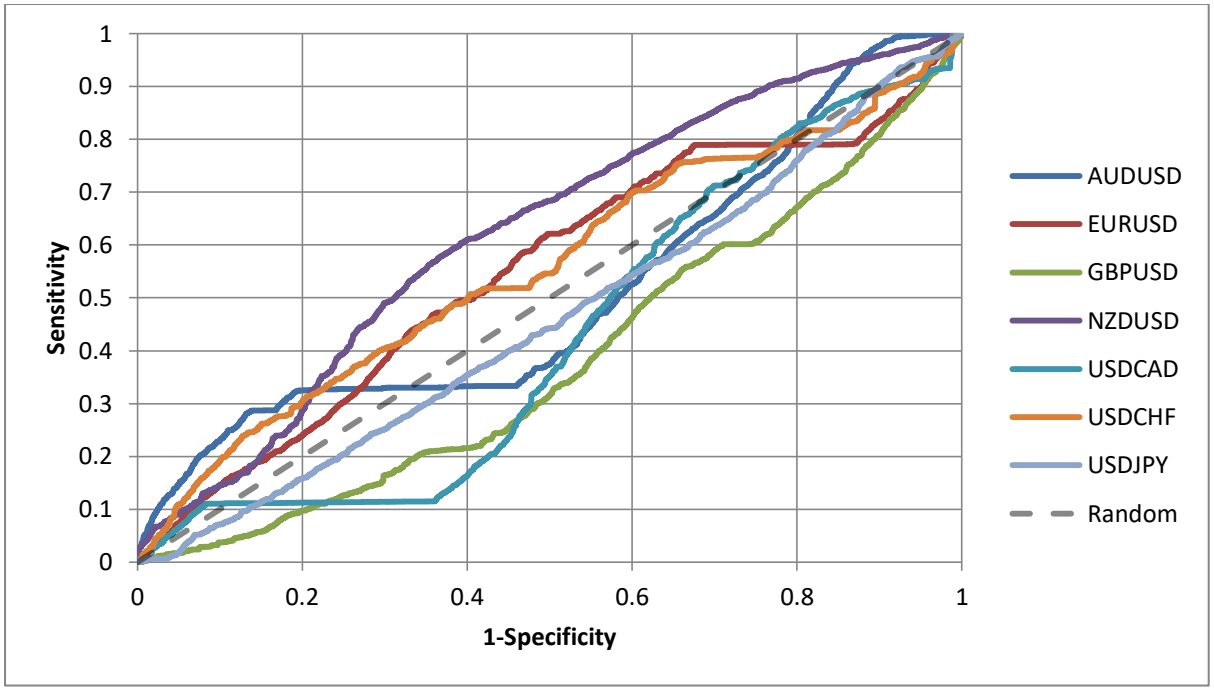


Figure 94: ROC curve of individual classifiers using validation data

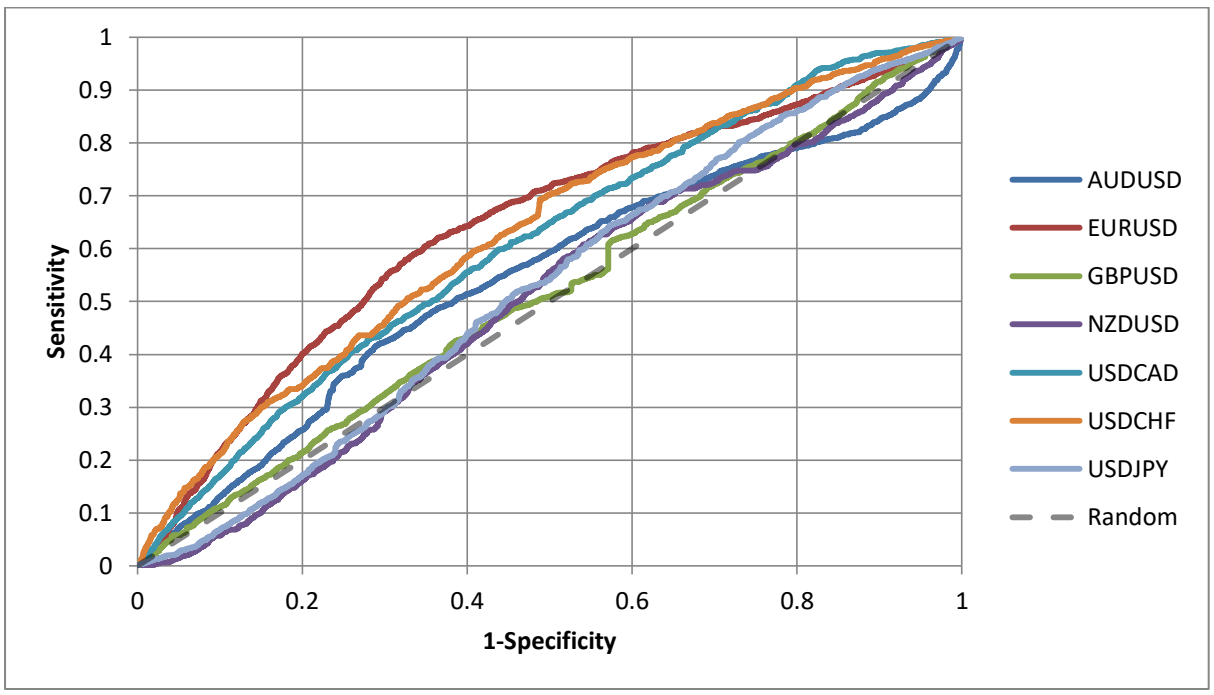


Figure 95: ROC curve of individual classifiers using outsample data

	AUDUSD	EURUSD	GBPUSD	NZDUSD	USDCAD	USDCHF	USDJPY	Average
Insample	0.831	0.857	0.889	0.873	0.833	0.946	0.836	0.867
Validation	0.510	0.549	0.383	0.623	0.429	0.557	0.459	0.502
Outsample	0.546	0.643	0.516	0.503	0.609	0.629	0.525	0.567

Table 22: Area under ROC curve for insample, validation and outsample data

8.3.2.3 Bootstrap Aggregation Classifier

Figure 96 shows the accuracy of bootstrap classifiers at each iteration of the Adaboost algorithm using the validation dataset. The bootstrap classifiers classified 29,653 technical analysis interpretations as profitable and 39,450 as unprofitable. Accuracy improves as the Adaboost algorithm iterates the EURUSD, USDJPY and GBPUSD data but decreases as it iterates the AUDUSD, NZDUSD and USDCAD data. Comparing the accuracy results of bootstrap classifiers with the accuracy results of individual Adaboost classifiers during the same period (Figure 91), the accuracy of the classifiers using USDCAD data dropped by 0.19 in both accuracy results. However, for the AUDUSD market data segment the individual Adaboost classifier's accuracy dropped by 0.2 but the accuracy did not drop in the bootstrap classifier. The accuracy of the bootstrap classifier using USDCHF data jumps up by 0.05 which did not occur in the individual Adaboost classifier. The accuracy results of the bootstrap classifiers using validation data are smoother than the individual classifiers.

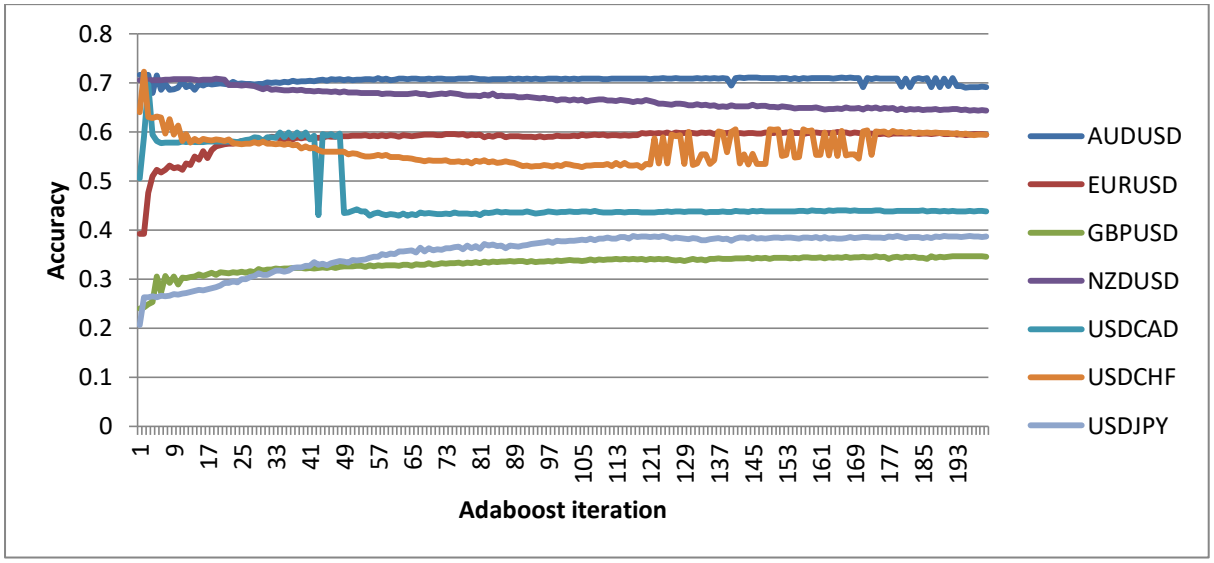


Figure 96: Accuracy of constructing bootstrap classifiers on validation data

Figure 97 shows the results from bootstrap classifiers at each iteration of the Adaboost algorithm using the outsample dataset. The bootstrap classifiers classified 23,078 technical analysis interpretations as profitable and 45,862 as unprofitable. There are no sharp accuracy drops, which were present in the validation dataset, suggesting that picking optimal classifiers before accuracy drops in the validation dataset is not beneficial. The accuracy values of all bootstrap classifiers converge by the 50th iteration, fluctuating only slightly in value. Using the outsample dataset, 5 out of the 7 bootstrap classifiers achieve

better than random performance at the final iteration of the Adaboost algorithm. The accuracy of the bootstrap classifier using USDJPY data is random (0.5) at the final iteration, and the bootstrap classifier using NZDUSD data is worse than random with a 0.45 accuracy.

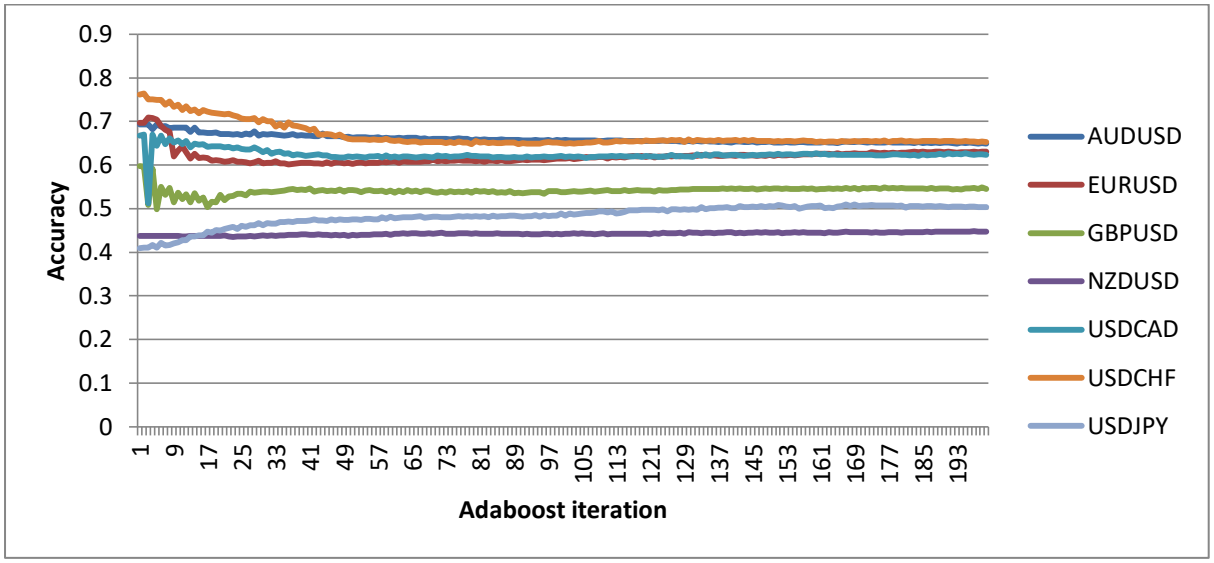


Figure 97: Accuracy of constructing bootstrap classifiers on outsample data

Figure 98 shows the average accuracy of bootstrap classifiers using the validation and outsample datasets at each iteration of the Adaboost algorithm. The accuracy values are very consistent except for the slight accuracy drop which was experienced in the bootstrap classifier using the USDCAD validation dataset segment.

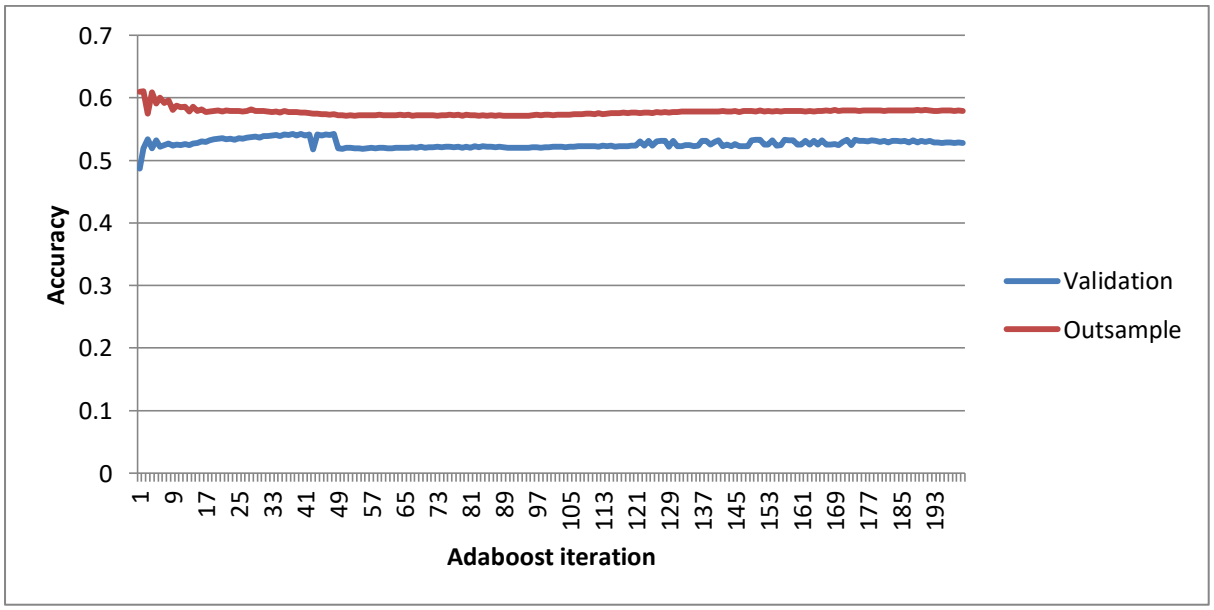


Figure 98: Average accuracy of constructing bootstrap classifiers on validation data and outsample data

The receiver operating characteristic (ROC) curve of bootstrap classifiers on validation data and outsample data can be seen in Figures 99 and 100. The bootstrap classifiers used on validation data separated technical analysis interpretations slightly better than random using NZDUSD and EURUSD data and performed randomly or worse on AUDUSD, GBPUSD, USDCAD, USDCHF and USDJPY data. However, bootstrap classifiers used on outsample data performed slightly better than random at separating the technical analysis interpretations (and at worst randomly). The area under the ROC curve (AUC) values for bootstrap classifiers using the insample, validation and outsample datasets are summarized in Table 23. The average AUC value on training data is 0.856, which indicates that the bootstrap classifiers have separated the technical analysis interpretations effectively for the training dataset. However, when the bootstrap classifiers are used on the validation and outsample dataset the AUC values are 0.491 and 0.563 respectively, indicating that the classifiers perform close to randomly or with a slight edge.

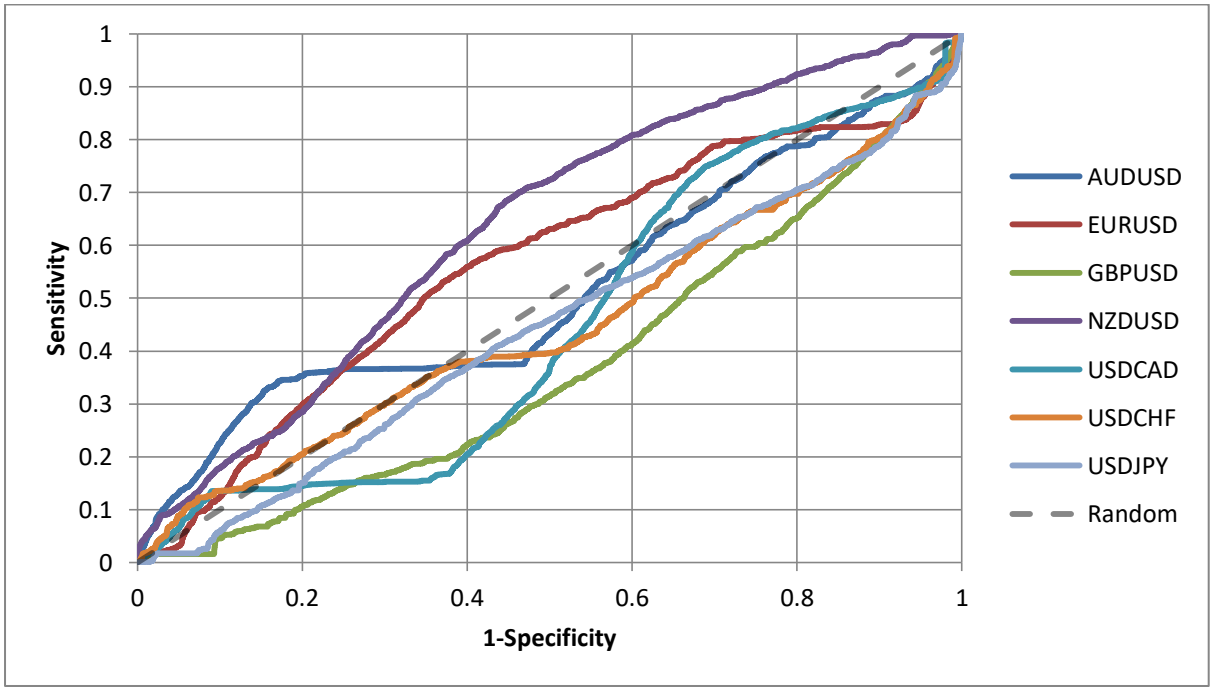


Figure 99: ROC curve of bootstrap classifiers using validation data

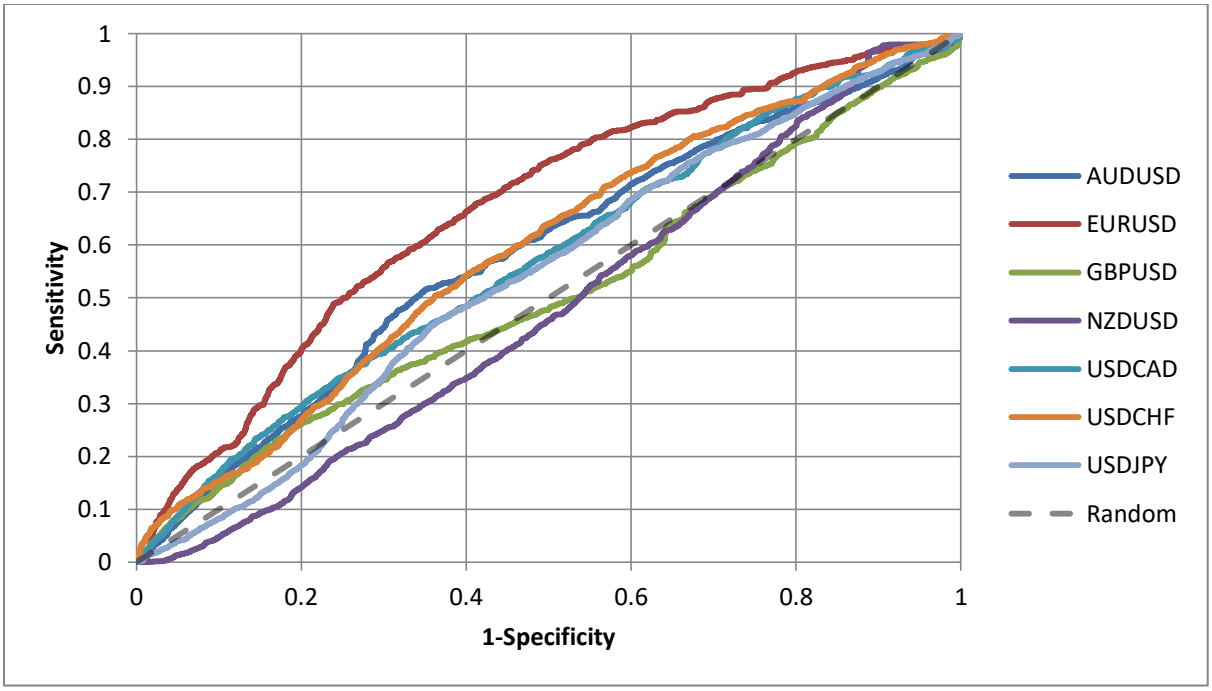


Figure 100: ROC curve of bootstrap classifiers using outsample data

	AUDUSD	EURUSD	GBPUSD	NZDUSD	USDCAD	USDCHF	USDJPY	Average
Insample	0.831	0.851	0.883	0.849	0.840	0.909	0.826	0.856
Validation	0.518	0.563	0.375	0.636	0.451	0.452	0.443	0.491
Outsample	0.582	0.669	0.508	0.479	0.571	0.591	0.541	0.563

Table 23: Area under ROC curve for insample, validation and outsample data

8.3.2.4 Bootstrap Aggregation and Feature-bagging Classifier

Figure 101 shows the accuracy of bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm using the validation dataset. The bootstrap and feature-bagging classifiers classified 26,261 technical analysis interpretations as profitable and 42,848 as unprofitable. The accuracy of the classifiers using EURUSD, USDJPY and GBPUSD data increase as the Adaboost algorithm iterates, but the accuracy decreases in the classifiers using AUDUSD, NZDUSD and USDCAD data. The bootstrap and feature-bagging classifiers using USDCHF data experienced frequent fluctuations in accuracy compared to the other classifiers during the first 100 iterations of the Adaboost algorithm. Comparing these results with the bootstrap classifiers on validation data in Figure 96, for both classifiers using USDCAD data the accuracy dropped which suggests overfitting. However, the overfitting

was overcome in the bootstrap and feature-bagging classifier soon after the classifier overfitted. The AUDUSD accuracy drop of 0.19 was not present in the bootstrap and feature-bagging classifier but was present in the individual classifier. The bootstrap classifier's accuracy (Figure 96) using USDCHF data fluctuated up and down by 0.05 until a final jump of 0.05 was observed. The bootstrap and feature-bagging classifier using USDCHF data experienced fluctuations at earlier iterations.

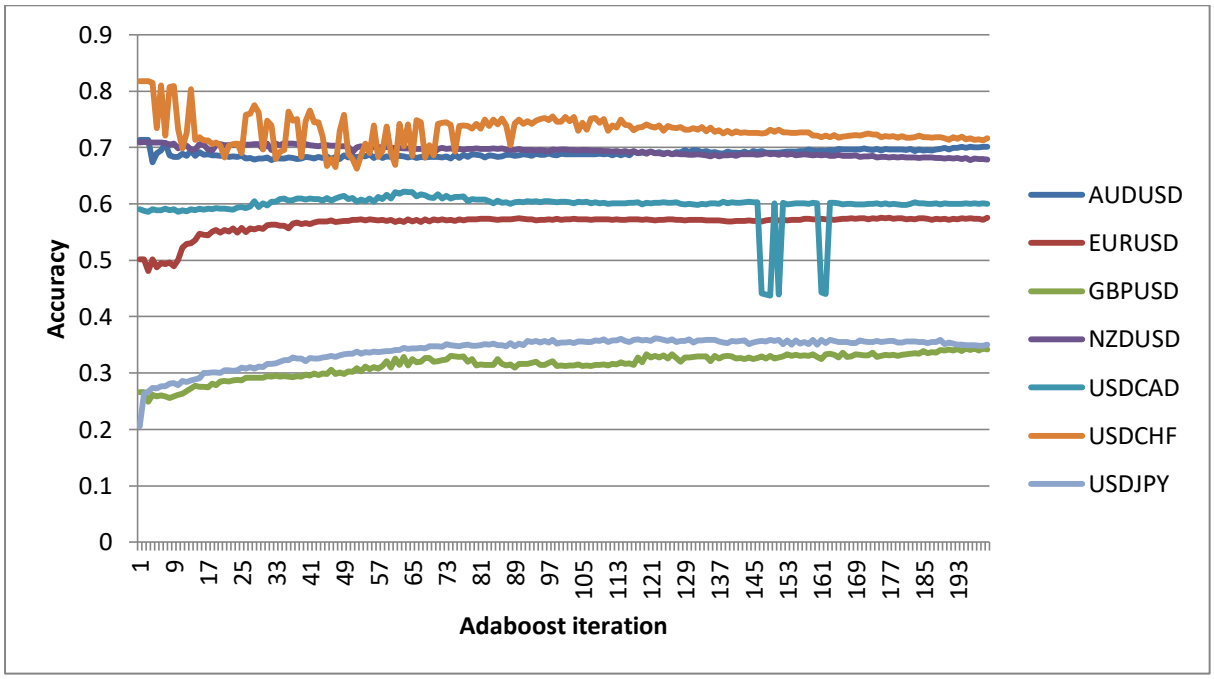


Figure 101: Accuracy of constructing classifiers with bootstrap & feature-bagging using validation data

Figure 102 shows the accuracy of the bootstrap and feature-bagging classifiers using outsample data. The bootstrap and feature-bagging classifiers classified 20,937 technical analysis interpretations as profitable and 48,016 as unprofitable. When using the outsample dataset, at the final iteration this classifier achieved better than random performance in 5 out of the 7 markets. The classifier using USDJPY data increased to an accuracy of 0.5 at the final iteration of the Adaboost algorithm from 0.42 initially. The classifier using NZDUSD data converged to an accuracy of 0.44. The accuracy results contain no sharp movements, which were evident in the validation dataset. The accuracy converges quickly, only slightly fluctuating in value when compared to the classifiers using the bootstrap aggregation technique on outsample data.

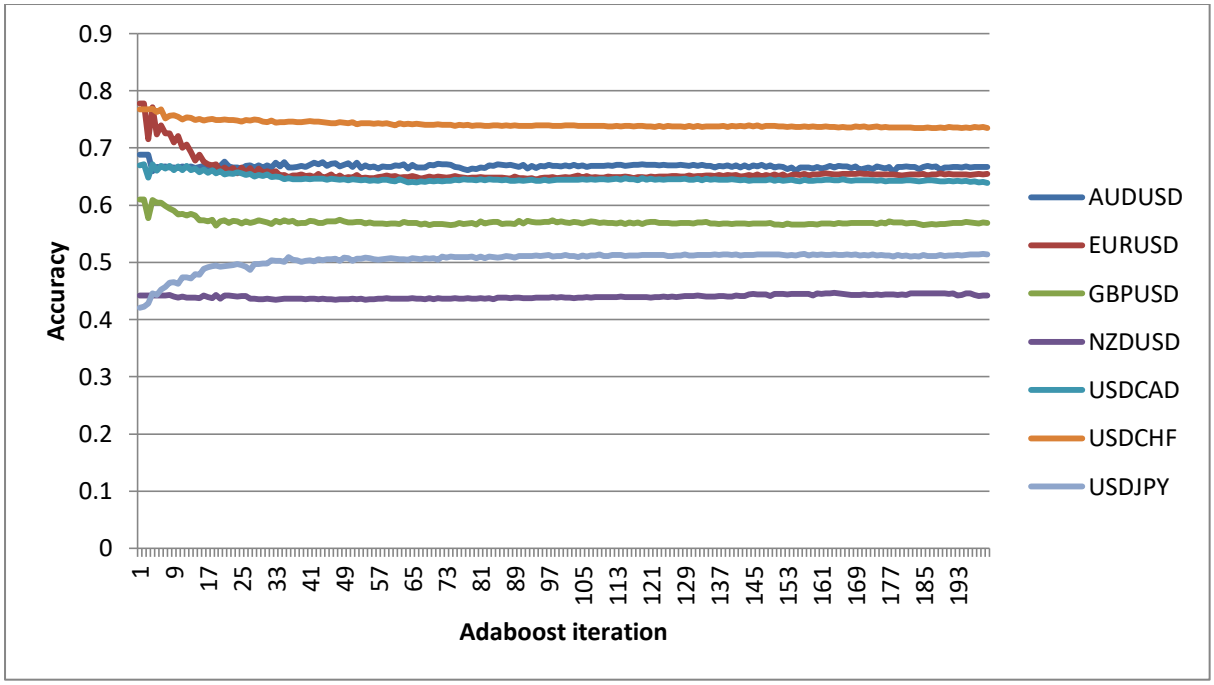


Figure 102: Accuracy of constructing classifiers with bootstrap & feature-bagging using outsample data

Figure 103 shows the average accuracy of bootstrap and feature-bagging classifiers using the validation and outsample datasets at each iteration of the Adaboost algorithm. The average accuracy of the bootstrap and feature-bagging classifiers on validation and outsample data is 0.57 and 0.6 respectively. The fluctuations experienced on USDCAD data using validation data are also observed in the average accuracy figure. No clear optimal iteration can be determined from the classifiers' accuracy on validation data. The accuracy values of classifiers using the outsample dataset converge within 30 iterations.

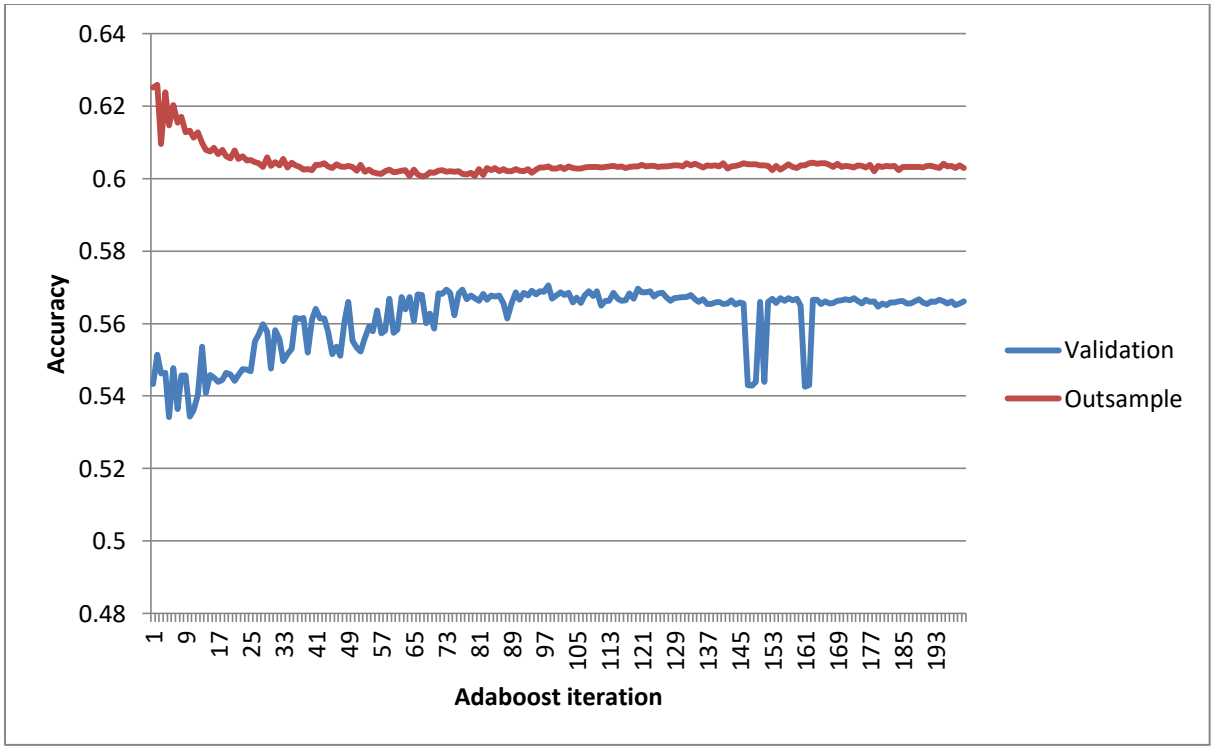


Figure 103: Average accuracy of constructing bootstrap and feature-bagging classifiers on validation data and outsample data

The ROC curve of bootstrap and feature-bagging classifiers on validation and outsample data can be seen in Figures 104 and 105. The classifiers used on validation data performed slightly better than random at separating technical analysis interpretations using AUDUSD, EURUSD and NZDUSD data, and performed randomly or worse on GBPUSD, USDCAD, USDCHF and USDJPY data. The AUDUSD classifier dipped to the random classifier line when $1 - specificity$ reached and surpassed the 0.35 value, which shows that the classifier is slightly better than random at classifying and identifying technical analysis interpretations when using a low classification threshold see Section 2.6. The classifier using EURUSD data performed worse than random when $1 - specificity$ reached and surpassed the 0.8 value.

The same bootstrap and feature-bagging classifiers used on outsample data performed slightly better. The classifiers using AUDUSD, EURUSD, USDCAD and USDCHF data separated the technical analysis interpretations slightly better than random, and classifiers using GBPUSD and NZDUSD data separated the technical analysis interpretations randomly. The classifier using USDJPY data performed slightly worse than random between the values of 0 and 0.25 on the x axis, then above the value of 0.25 the classifier improved to slightly better than random. The AUC values for the bootstrap and feature-bagging classifiers using the

insample, validation and outsample datasets are summarized in Table 24. The average AUC value on training data is 0.842, which indicates that the classifiers have separated the technical analysis interpretations according to the training set effectively. Similarly to the bootstrap classifiers, the classifiers using validation and outsample data have AUC values of 0.506 and 0.578 respectively. The classifiers perform randomly when using validation data and with a slight edge when using outsample data.

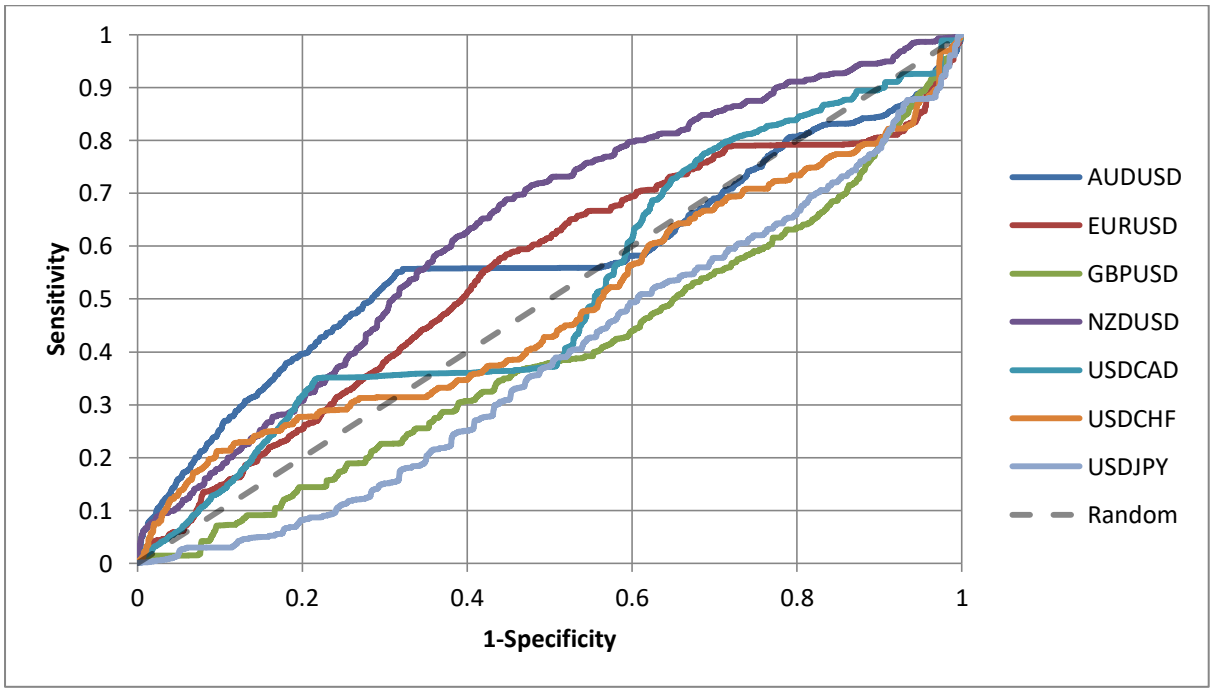


Figure 104: ROC curve of bootstrap & feature-bagging classifiers using validation data

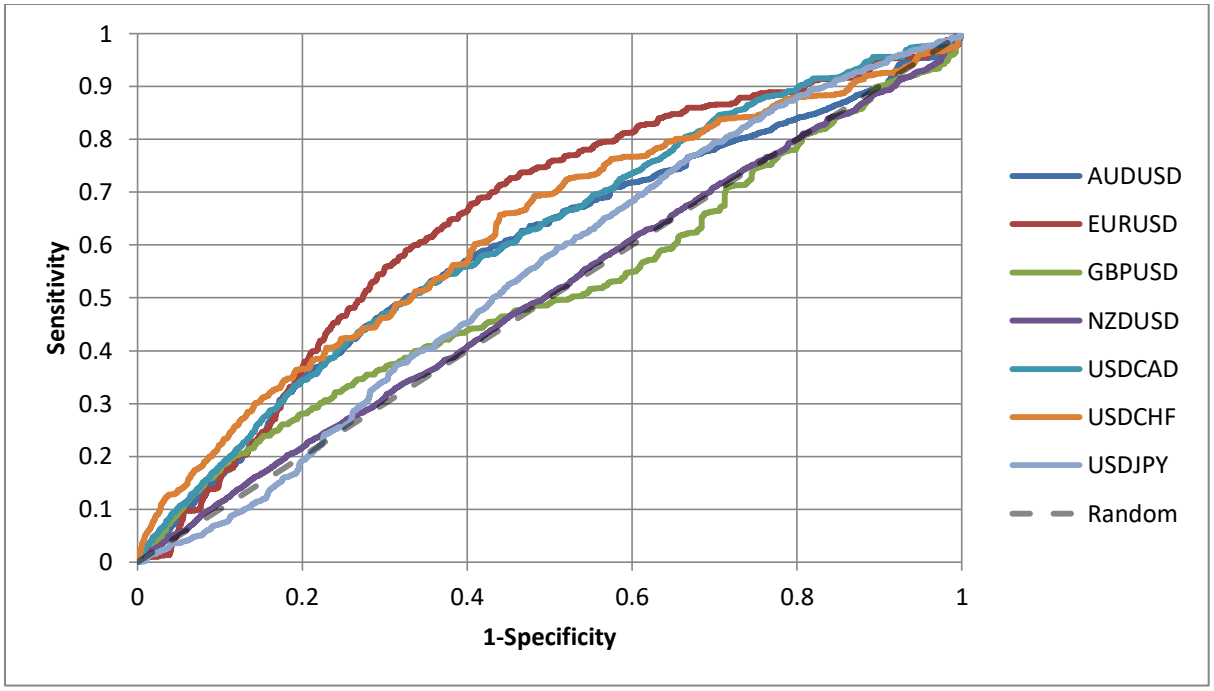


Figure 105: ROC curve of bootstrap & feature-bagging classifiers using outsample data

	AUDUSD	EURUSD	GBPUSD	NZDUSD	USDCAD	USDCHF	USDJPY	Average
Insample	0.819	0.844	0.881	0.827	0.827	0.888	0.809	0.842
Validation	0.572	0.546	0.398	0.635	0.519	0.484	0.388	0.506
Outsample	0.592	0.650	0.514	0.505	0.613	0.625	0.544	0.578

Table 24: Area under ROC curve for insample, validation and outsample data

8.3.2.5 Summary of Accuracy Analyses

The accuracy and ROC results given above summarise the performance of each classification system. One of the purposes of these results is to determine whether an optimal iteration during the Adaboost algorithm can be chosen using the validation dataset, which can then be used to predict profitable and unprofitable technical analysis interpretations on unseen outsample data. The results showed overfitting in some classifiers using the validation dataset. However, no evident overfitting occurred in the outsample dataset. On outsample data, classifiers which had overfitted on validation data operated as effectively as classifiers which did not overfit on validation data. Ignoring overfitting, the results obtained from classifiers using the validation and outsample dataset showed that within 50 to 100 iterations, the accuracy of the classifiers stabilised. The classification systems between the 50^{th} and 100^{th} iteration may provide the best iterations for choosing the optimal classification system, as the accuracy has converged to a value (low bias) and additional

iterations would risk increasing the variance of the classifier, see Section 2.3. This strategy for choosing the optimal Adaboost algorithm iteration is a good trade-off for bias and variance. Additional iterations of the Adaboost algorithm did not change or substantially benefit the classifier's accuracy performance. Also, picking an optimal Adaboost iteration before overfitting has occurred may be unreliable as no overfitting was seen in the outsample dataset.

The results may have been different if the datasets were created using different time periods. However, the results produced are sufficient to warrant further investigation into Adaboost classifier construction for improved models of traders.

The bootstrap classifiers and the bootstrap and feature-bagging classifiers had accuracy results that were smoother than individual Adaboost classifier results but this was expected as these techniques average the classification of 51 individual Adaboost classifiers.

The ROC curve results showed that the general performance of the classifiers were either random or had a slight advantage at separating unprofitable and profitable technical analysis interpretations on outsample data.

The bootstrap and feature-bagging classifiers performed better than the individual classifiers and the bootstrap classifiers, reducing the overfitting that was observed in the other classification systems on the validation data.

8.3.3 Classifier Precision and Recall Performance

This section will report the precision and recall of classification systems at each iteration of the Adaboost algorithm during training. Then report validation and outsample performance for the individual classifier, the bootstrap aggregation classification system, and bootstrap aggregation with feature bagging classification system.

8.3.3.1 Training Dataset Results

Figures 106, 108 and 110 show the precision values, and Figures 107, 109 and 111 show the recall values, of the constructing classifiers at each iteration of the Adaboost algorithm using the training dataset. The training dataset is used to create the classifier. Each graph reports seven separate results derived from each of the foreign exchange markets.

Figures 106 and 107 show the results of individual classifiers and, as expected, the precision and recall generally increases with each iteration of the Adaboost algorithm. An abnormal change in precision and recall is observed at the 16th iteration of the Adaboost algorithm. At each iteration, the weights of each technical analysis interpretation during the training phase is adjusted, which adds more weight to technical analysis interpretations that are incorrectly categorized. The abnormality may be caused by enormous weight values that arise from technical analysis interpretations that are consistently misclassified. The accuracy of these individual classifiers (Figure 85), however, experienced no abnormal drop or increase at iteration 16.

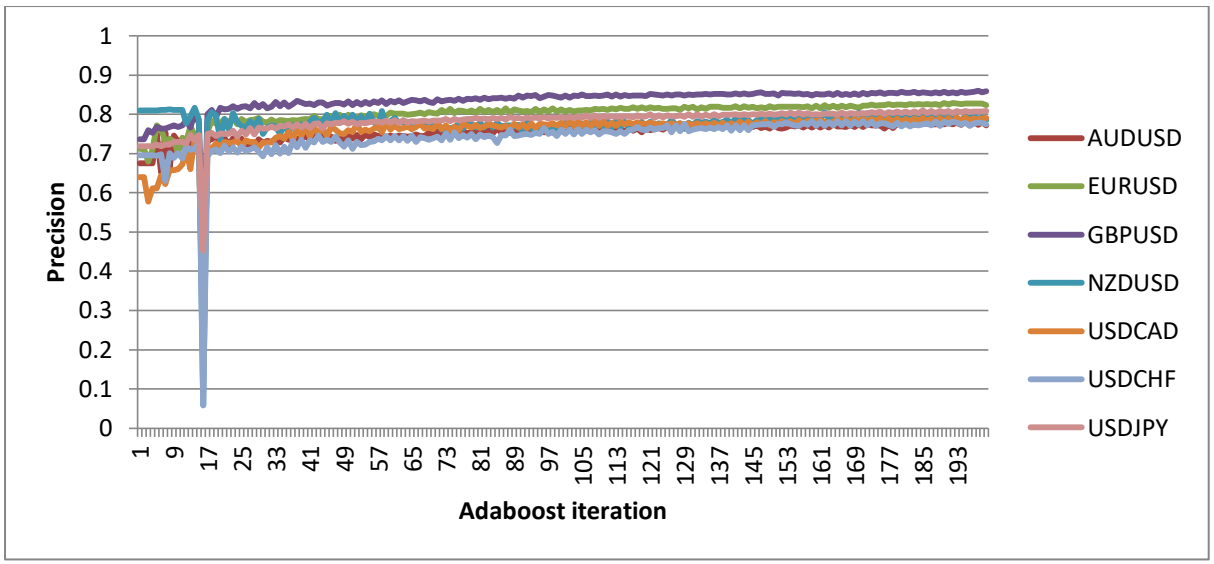


Figure 106: Precision of individual classifiers during training

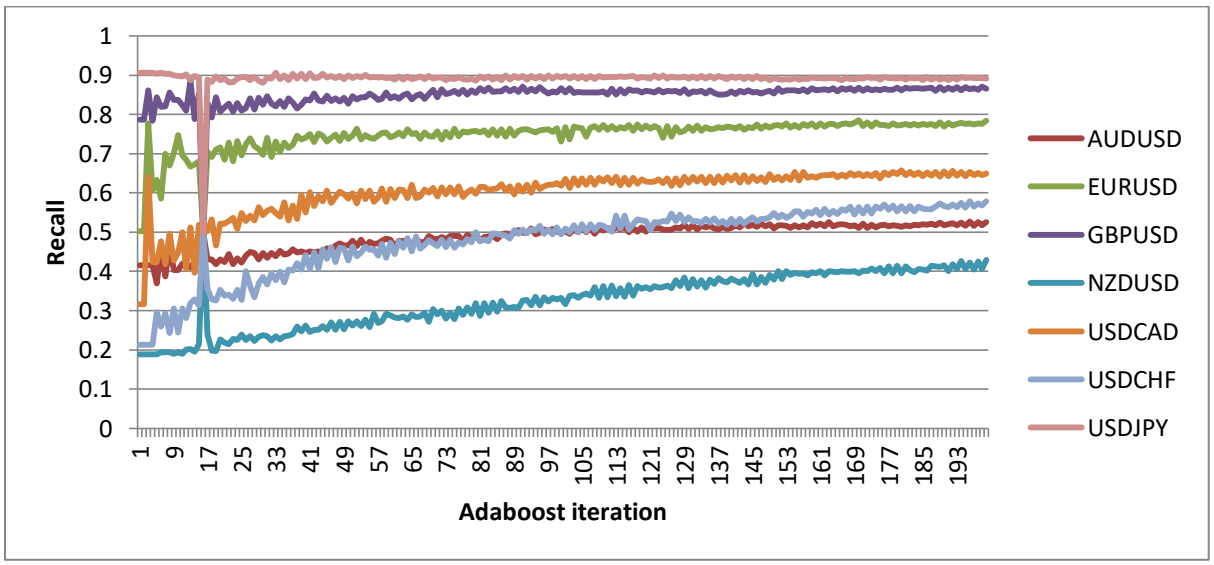


Figure 107: Recall of individual classifiers during training

Figures 108 and 109 show the precision and recall results of bootstrap classifiers. As expected, the values generally increase with each iteration of the Adaboost algorithm and are smoother than the individual classifier results. The abnormalities at the 16th iteration are also present in these results.

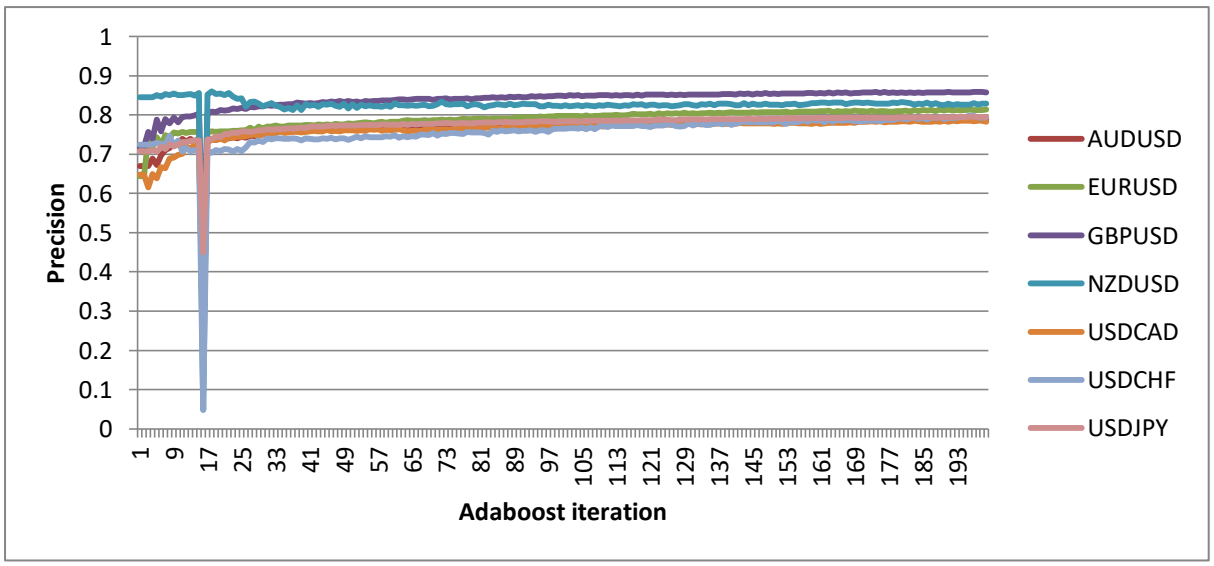


Figure 108: Precision of bootstrap classifiers during training

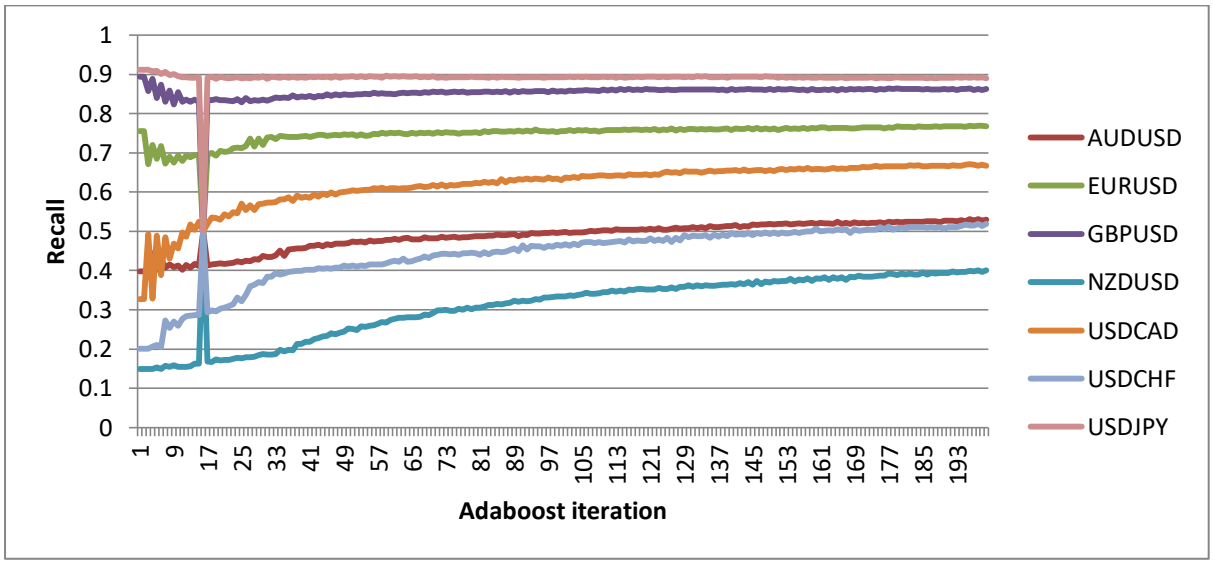


Figure 109: Recall of bootstrap classifiers during training

Figures 110 and 111 show the precision and recall results of bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm. This classification system shows similar results to those for the bootstrap classifiers. The precision and recall results generally increase at each iteration of the Adaboost algorithm. Compared to the individual classifiers,

the results are smoother but contain the same abnormality at the 16th iteration. As was the case for accuracy (Section 8.3.2.1), this measure supports the view that the classifiers have learnt the training data.

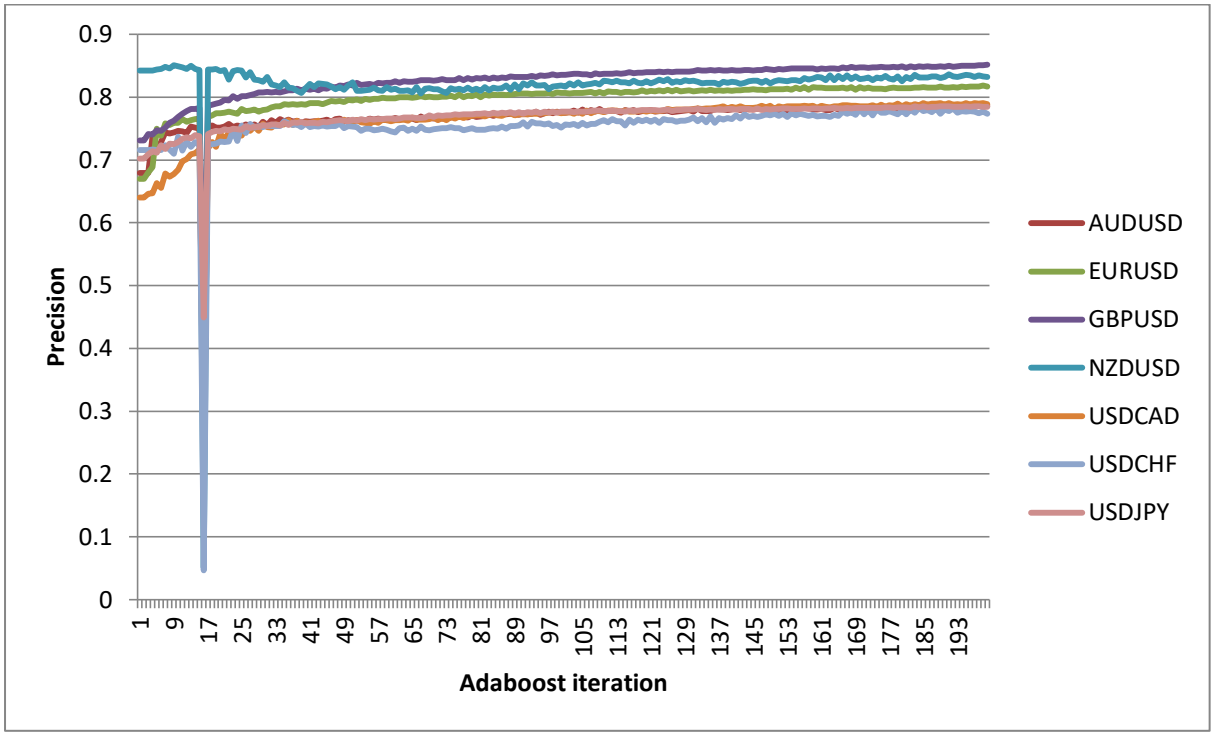


Figure 110: Precision of bootstrap and feature-bagging classifiers during training

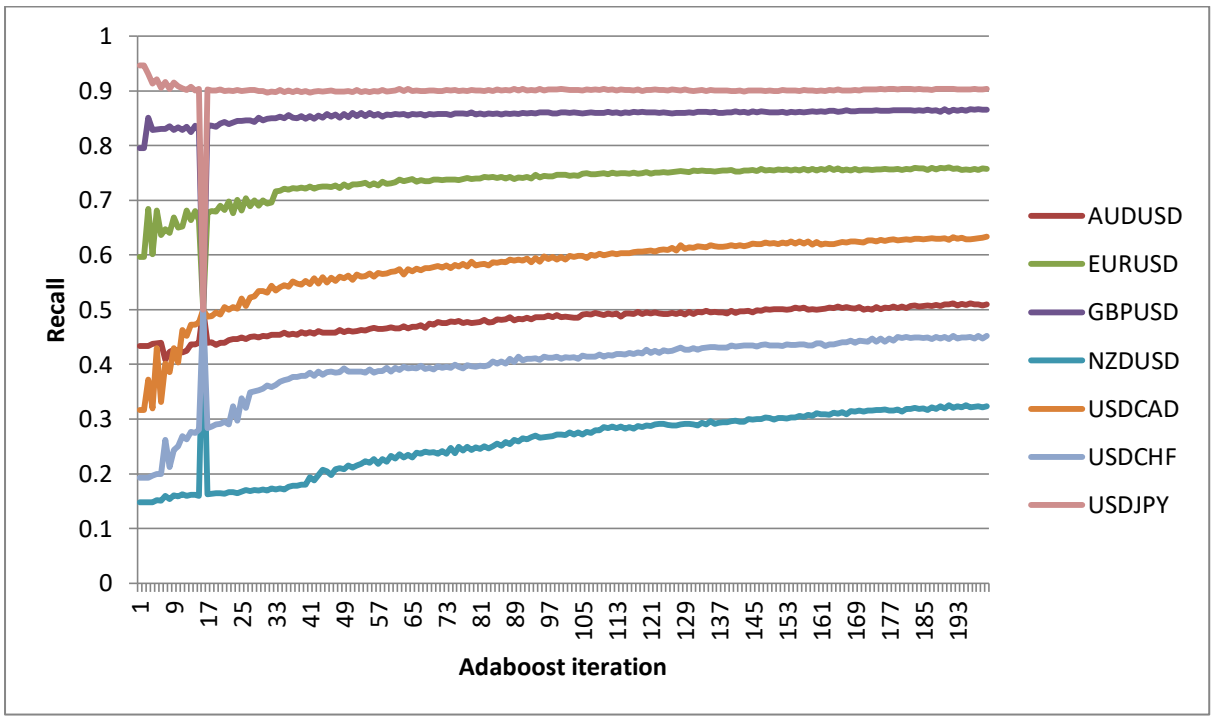


Figure 111: Recall bootstrap and feature-bagging classifiers during training

8.3.3.2 Individual Classifier

Figure 112 shows the precision values of individual classifiers at each iteration of the Adaboost algorithm using validation data. The results are similar to those for accuracy (Figure 91) and in particular, the precision of individual classifiers using AUDUSD and USDCAD data also dropped significantly by roughly 0.2. The precision results of classifiers at the 200th iteration of the Adaboost algorithm achieved a precision value of less than 0.4. Prior to the individual classifier using USDCAD data overfitting, the classifier had a precision of 0.5.

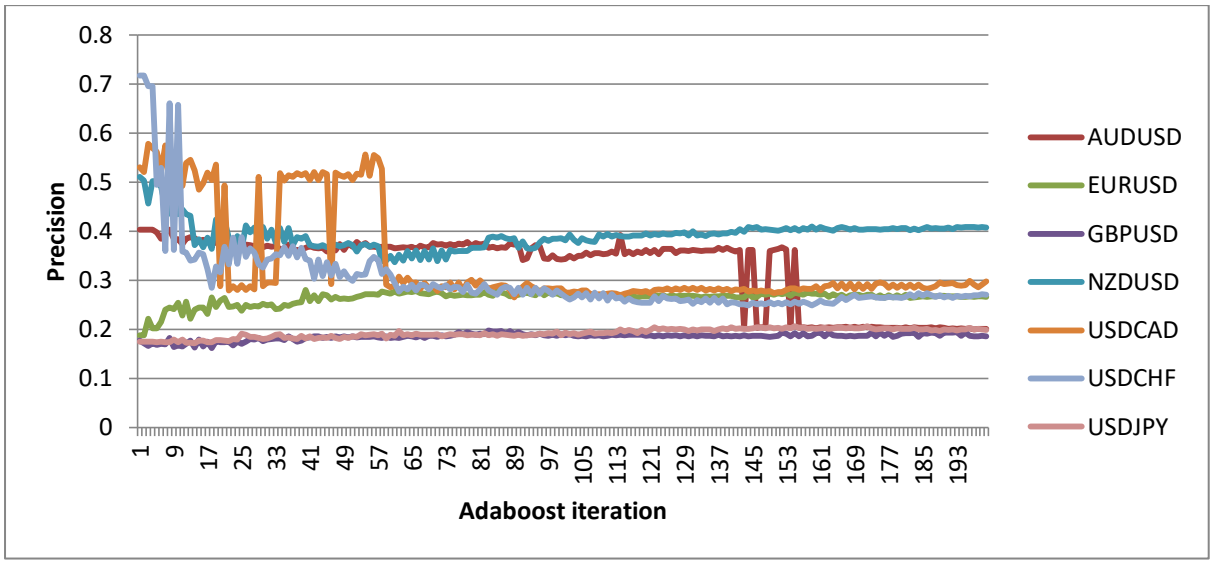


Figure 112: Precision of constructing individual classifiers on validation data

Figure 113 shows the recall values of individual classifiers at each iteration of the Adaboost algorithm using validation data. The recall value of the classifier using NZDUSD data increases as Adaboost iterates, but for the other markets the classifiers converge within 50 iterations. The best recall values at the 200th iteration of the Adaboost algorithm are 0.61, 0.58 and 0.55 for classifiers using GBPUSD, USDJPY and EURUSD respectively; the worst recall values are 0.28, 0.35 and 0.35 for classifiers using USDCAD, USDCHF and AUDUSD data. The recall value for the classifier using NZDUSD data is 0.49, i.e. close to random.

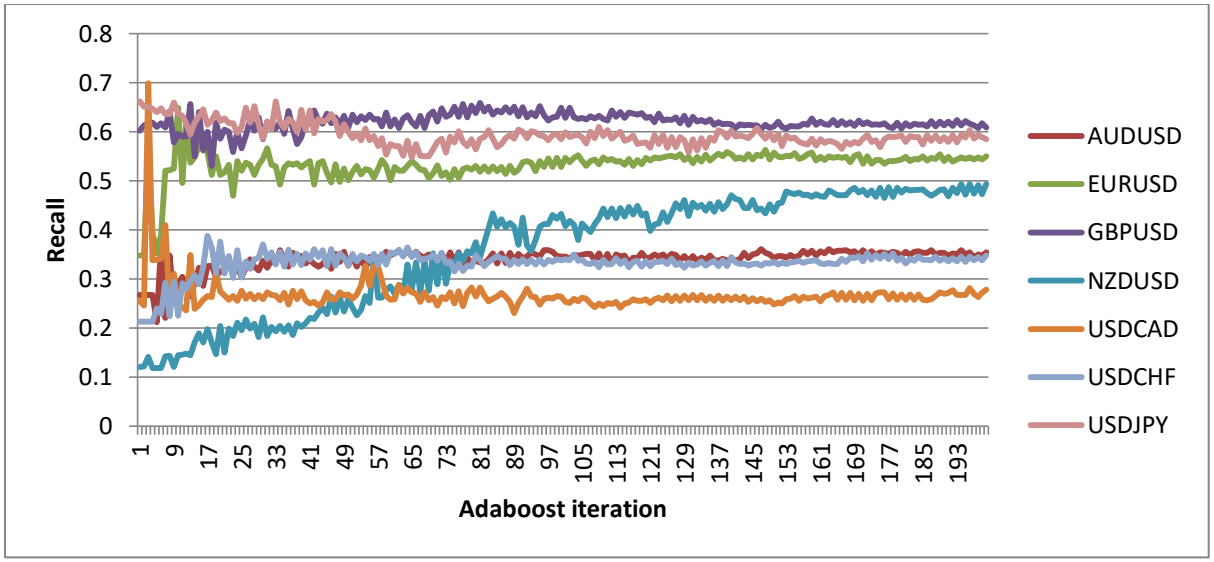


Figure 113: Recall of constructing individual classifiers on validation data

Figure 114 shows the precision values of individual classifiers at each iteration of the Adaboost algorithm using outsample data. While the precision values of the individual classifier using NZDUSD data increased at each iteration, those of the other classifiers decreased or remained the same. When comparing the precision values of validation and outsample data, there is no obvious choice of a classifier for optimal future precision performance from the validation dataset results. For outsample data, both the precision and recall values (Figure 115) of the classifier using NZDUSD validation data increased which is not the case for any of the other markets. Except for the individual classifier using NZDUSD data on validation data, the recall values of individual classifiers using the validation and outsample dataset converge within 50 iterations. No optimal Adaboost iteration can be determined from the recall values obtained from the validation dataset. Interestingly, the top performing recall classifiers on validation data (GBPUSD, USDJPY and EURUSD data) were also the top three performing classifiers for recall on outsample data.

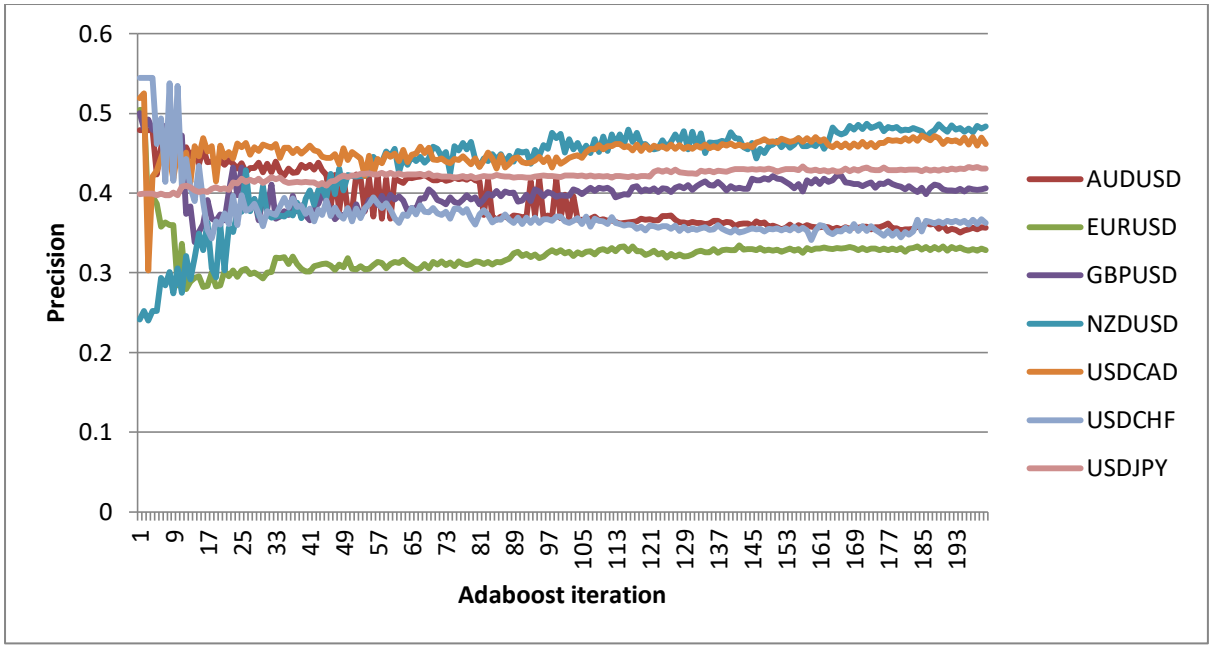


Figure 114: Precision of constructing individual classifiers on outsample data

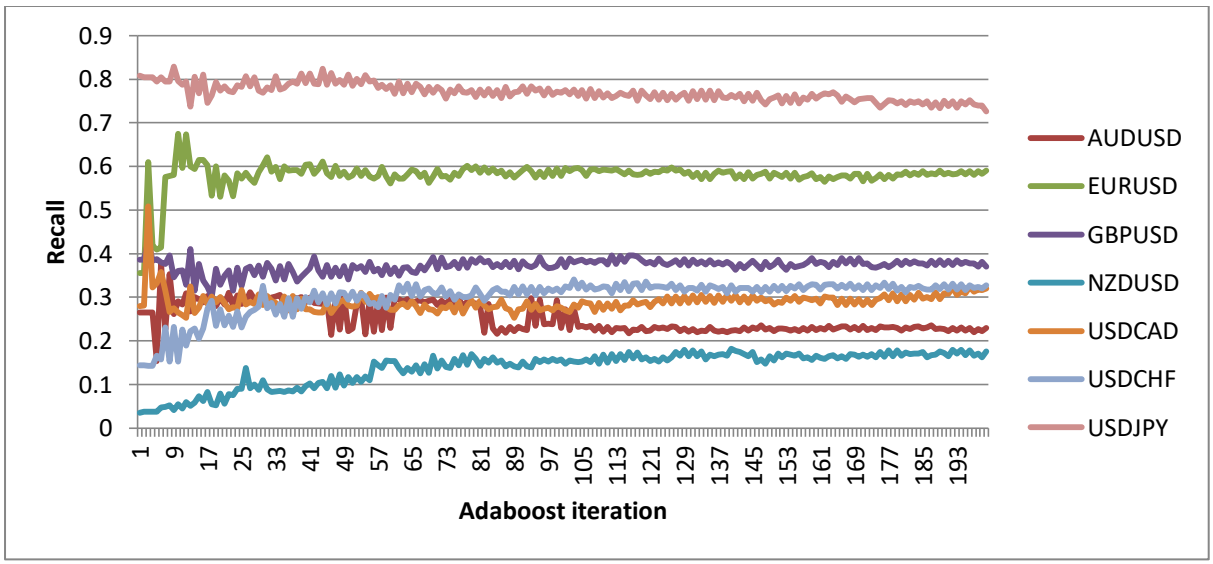


Figure 115: Recall of constructing individual classifiers on outsample data

Figures 116 and 117 show the average precision and recall values at each iteration of the Adaboost algorithm using validation and outsample data. The average precision of classifiers at the 200th iteration of the Adaboost algorithm on validation and outsample data is 0.26 and 0.40 respectively. The average precision for the individual classifiers using AUDUSD and USDCAD validation data is about 0.32 prior to overfitting. The precision performance of individual classifiers is worse than random at identifying profitable technical analysis

algorithms. The average recall of classifiers at the 200th iteration of the Adaboost algorithm on validation data is 0.46 and on outsample data 0.39.

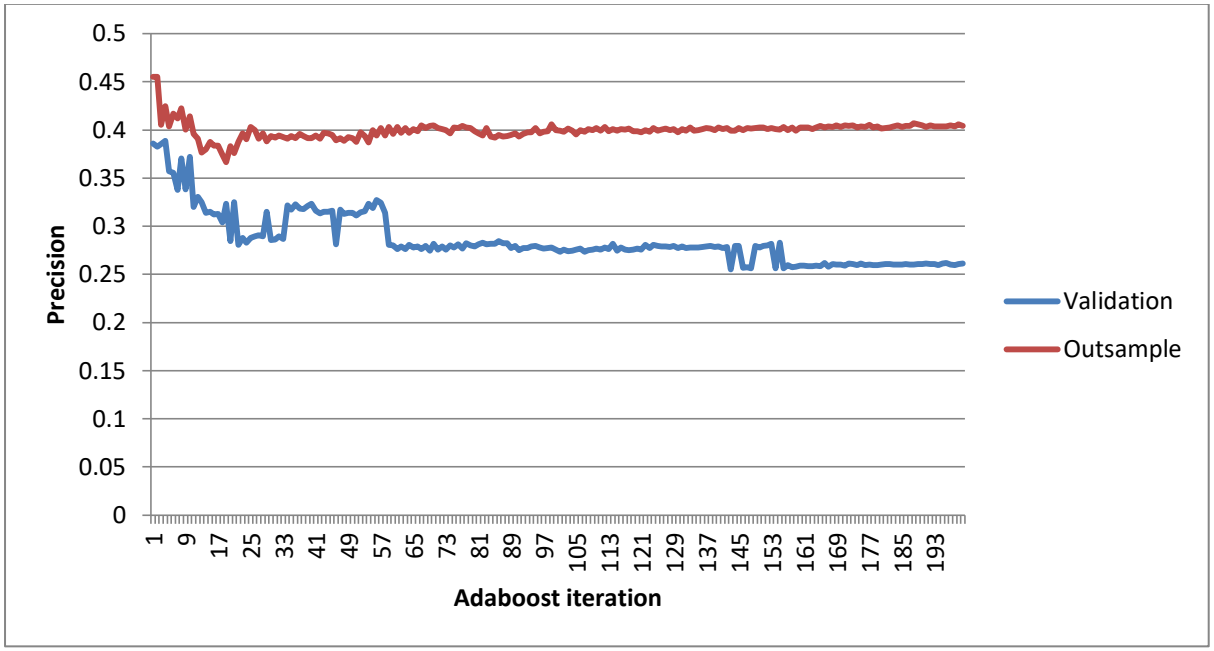


Figure 116: Precision of constructing classifiers on validation data and outsample data

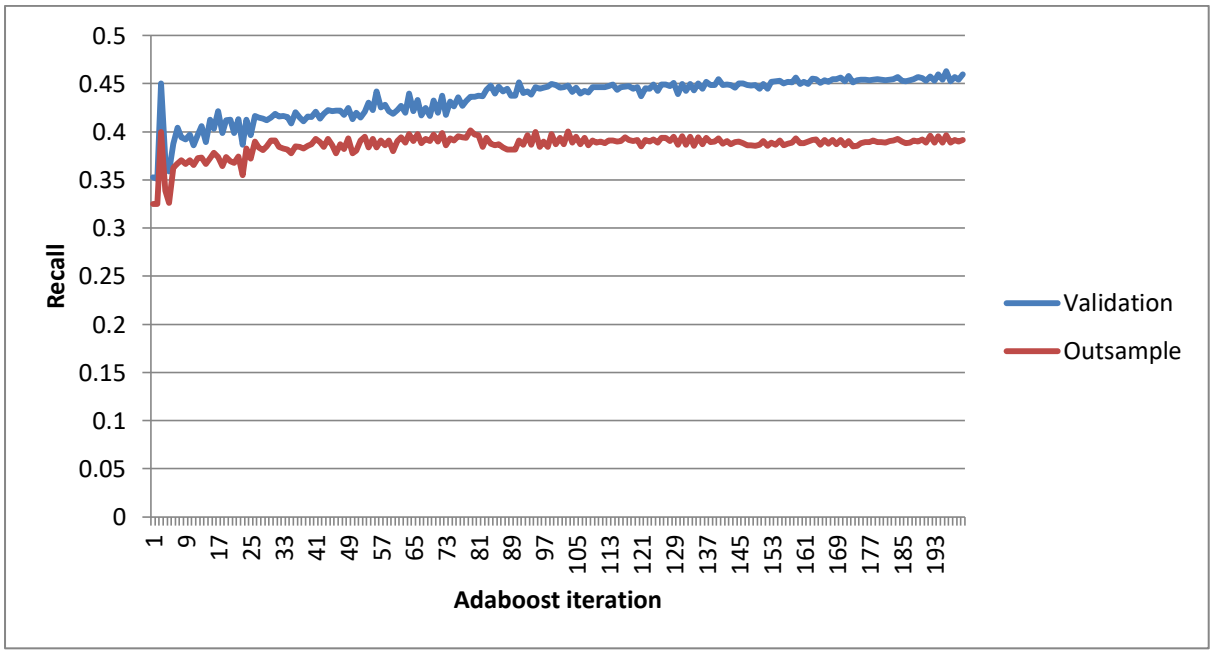


Figure 117: Recall of constructing classifiers on validation data and outsample data

8.3.3.3 Bootstrap Aggregation Classifier

Figure 118 shows the precision values of bootstrap classifiers at each iteration of the Adaboost algorithm using validation data. The precision values increase using EURUSD data, and there is also a slight increase using AUDUSD data before dropping at the 193rd iteration. The precision of the bootstrap classifiers using GBPUSD, USDCHF and USDJPY data show only slight variation, but for the classifier using GBPUSD data jumped up 0.03. A significant precision drop of 0.2 at iteration 47 is observed in the USDCAD market.

Comparing the precision results of bootstrap classifiers with the precision results of individual classifiers (Figure 112), both show a drop of 0.2 for USDCAD data. However, the precision drop of 0.2 in the individual classifier using AUDUSD data is not present in the corresponding bootstrap classifier. The precision of the bootstrap classifier using USDCHF data increased slightly from 0.17 to 0.2. The individual classifier using the same USDCHF data interestingly has a high precision value of 0.72 at iteration 1 of the Adaboost algorithm, decreasing to 0.27. The individual classifier outperformed the bootstrap classifier using USDCHF data. The precision results of individual classifiers and bootstrap classifiers using EURUSD, GBPUSD, NZDUSD, USDCAD and USDJPY data behaved the same. The precision values were smoother in the bootstrap classifier results than in the individual classifier results.

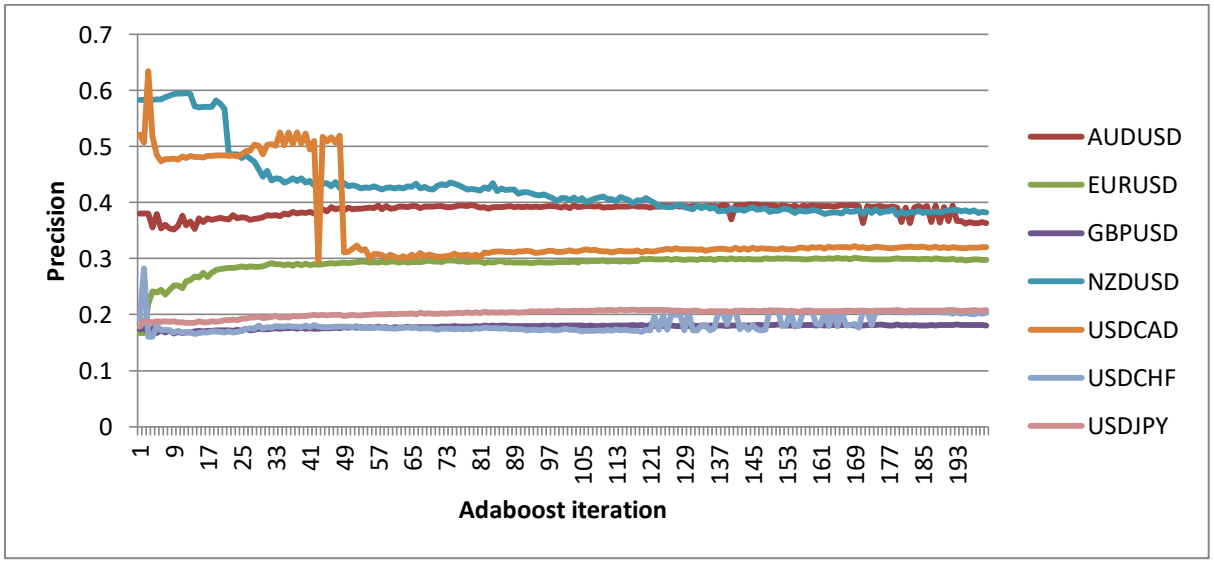


Figure 118: Precision of constructing classifiers using bootstrap on validation data

Figure 119 shows the recall values of the bootstrap classifiers at each iteration of the Adaboost algorithm using validation data. The sharp changes seen in the precision values are not observed in these results. The recall values of the bootstrap classifiers using NZDUSD, USDCAD and USDCHF data increase as Adaboost iterates. For AUDUSD, EURUSD and GBPUSD, the recall values quickly converge within 10 iterations. The recall values of the bootstrap classifier using USDJPY data decrease slightly before converging, but this classifier has the highest recall value.

Comparing the bootstrap classifier recall results with the individual classifier recall results (Figure 113), the recall values of the individual classifier using the NZDUSD market data increased from 0.12 at iteration 1 to 0.49 at iteration 200, while those of the bootstrap classifier rose from 0.09 to 0.32. The individual and bootstrap classifiers using the same foreign exchange market data behaved in largely the same way, but the recall values of the bootstrap classifiers are smoother than those of the individual classifiers.

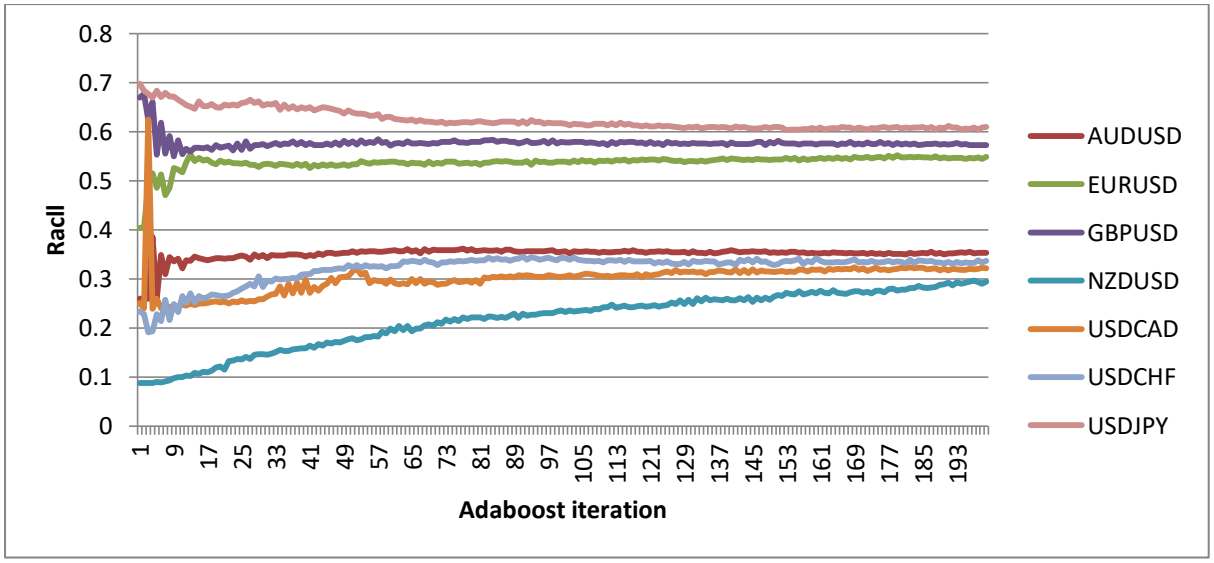


Figure 119: Recall of constructing classifiers using bootstrap on validation data

Figure 120 shows the precision values of bootstrap classifiers at each iteration of the Adaboost algorithm using outsample data. None of the sharp movements in precision evident for the validation dataset are observed here. The precision values of the bootstrap classifier using NZDUSD data increase with iteration, and the recall values increase slightly for the bootstrap classifier using USDJPY data. The precision decreases in classifiers using

AUDUSD, USDCAD and USDCHF data, and converge after just a few iterations in the EURUSD, GBPUSD and USDCAD.

Comparing the bootstrap classifier precision results with the precision results of individual classifiers (Figure 114), similar increases, decreases and convergences of precision values are observed. The precision values at the 200th iteration of the Adaboost algorithm are slightly lower in the bootstrap aggregation classifiers using AUDUSD, NZDUSD, USDCAD and USDCHF data whilst the other precision values are the same.

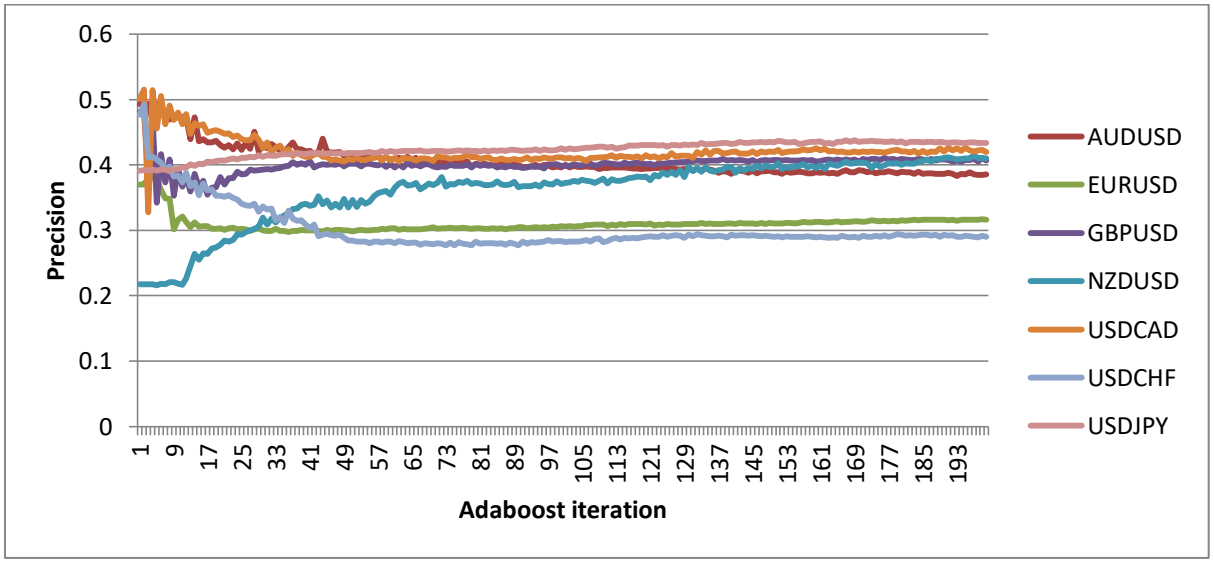


Figure 120: Precision of constructing classifiers using bootstrap on outsample data

Figure 121 shows the recall values of bootstrap classifiers at each iteration of the Adaboost algorithm using outsample data. The three top performing bootstrap classifiers on validation data, i.e. those using EURUSD, GBPUSD and USDJPY data, are also the three top performing bootstrap classifiers on outsample data. The recalls of the bootstrap classifiers using NZDUSD and USDCHF data increase slowly, whereas the recall values of the bootstrap classifiers using USDJPY decrease slowly. The recall values of the other bootstrap classifiers quickly converge.

Comparing the recall results of the bootstrap classifiers with the recall results of the individual classifiers (Figure 115), the recall values converge towards the same values except for the bootstrap classifier using NZDUSD data which is 0.08 lower. The precision and recall values of the bootstrap classifiers are smoother than those of the individual classifiers.

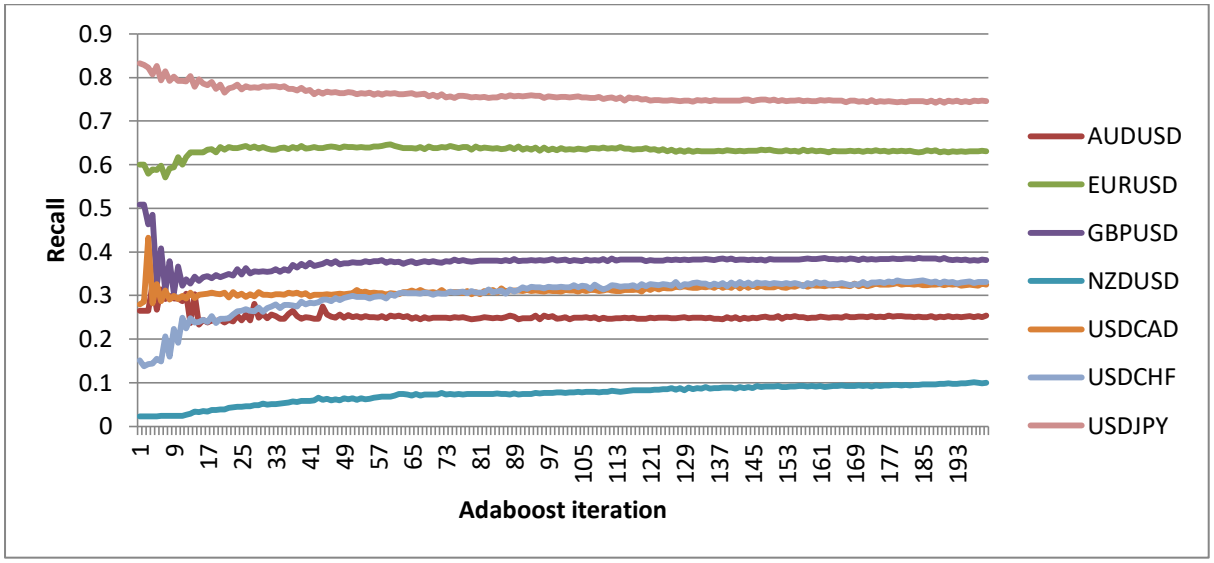


Figure 121: Recall of constructing classifiers using bootstrap on outsample data

Figures 122 and 123 show the average precision and recall values at each iteration of the Adaboost algorithm using validation and outsample data. The average precision value of bootstrap classifiers using validation and outsample data converges within 25 iterations. However, the overfitting of the bootstrap classifier using USDCAD data at the 46th iteration results in a precision drop in the validation dataset. The average recall values of the bootstrap classifiers using validation and outsample data increase steadily and converge.

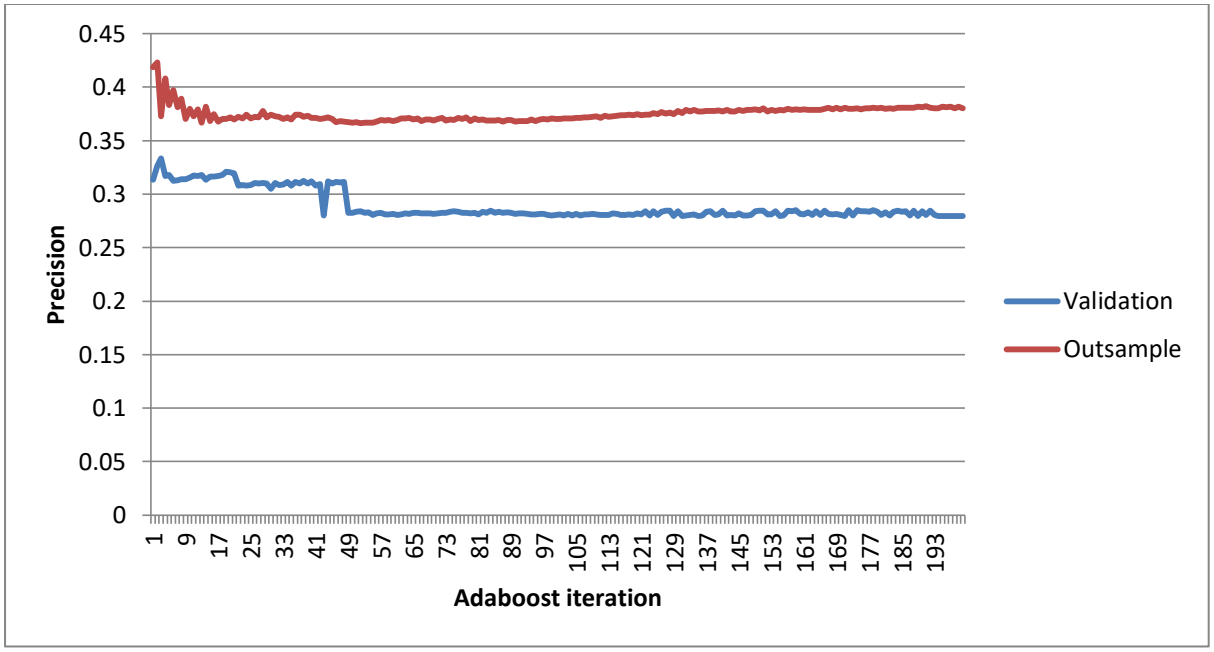


Figure 122: Average precision of constructing bootstrap classifiers on validation data and outsample data

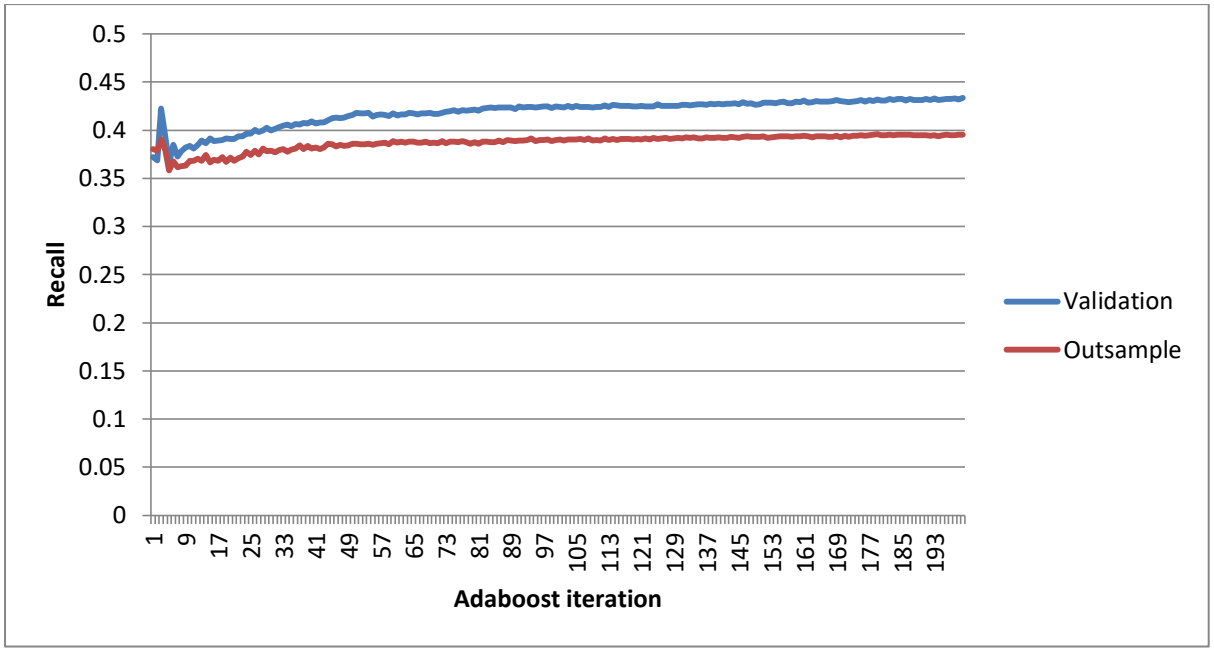


Figure 123: Average recall of constructing bootstrap classifiers on validation data and outsample data

8.3.3.4 Bootstrap Aggregation and Feature-bagging Classifier

Figure 124 shows the precision values of bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm using validation data. The precision values of classifiers

using AUDUSD, GBPUSD, USDCAD and USDJPY quickly converge within a few iterations. The precision of the classifier using USDCAD data shows a drop in precision but recovers around the 150th iteration, suggesting that the classifier recovers from overfitting. The bootstrap and feature-bagging classifiers avoided the instances of overfitting that occurred in the other classification systems.

The precision of the bootstrap and feature-bagging classifier using USDCHF data oscillates frequently in value decreasing from an initial precision value of roughly 0.75 to 0.3. The classifiers using EURUSD, GBPUSD, NZDUSD and USDJPY data all behaved the same as for previous classification systems.

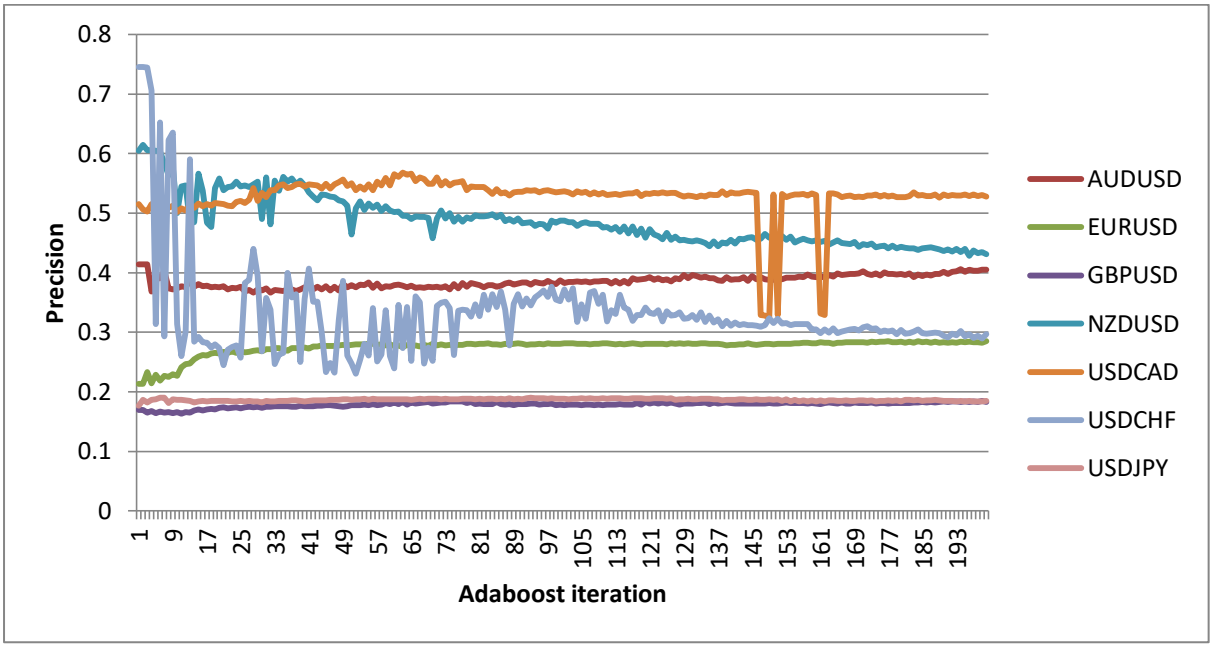


Figure 124: Precision of constructing classifiers in bootstrap with feature-bagging on validation data

Figure 125 shows the recall values of the bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm using validation data. The recall values behave in the same way as for the bootstrap classifiers. The recall values of the bootstrap and feature-bagging classifiers using NZDUSD and USDCHF data perform worse at iteration 200 compared to the bootstrap classifiers (Figure 119). The recall values of the classifier using USDCHF data converge quickly whereas those of the classifier using NZDUSD data increase steadily. The recall values of the classifier using USDCAD data increase to an optimal recall

value at iteration 60, before decreasing. This indicates that the classifier at iteration 60 should perform the best on USDCAD outsample data.

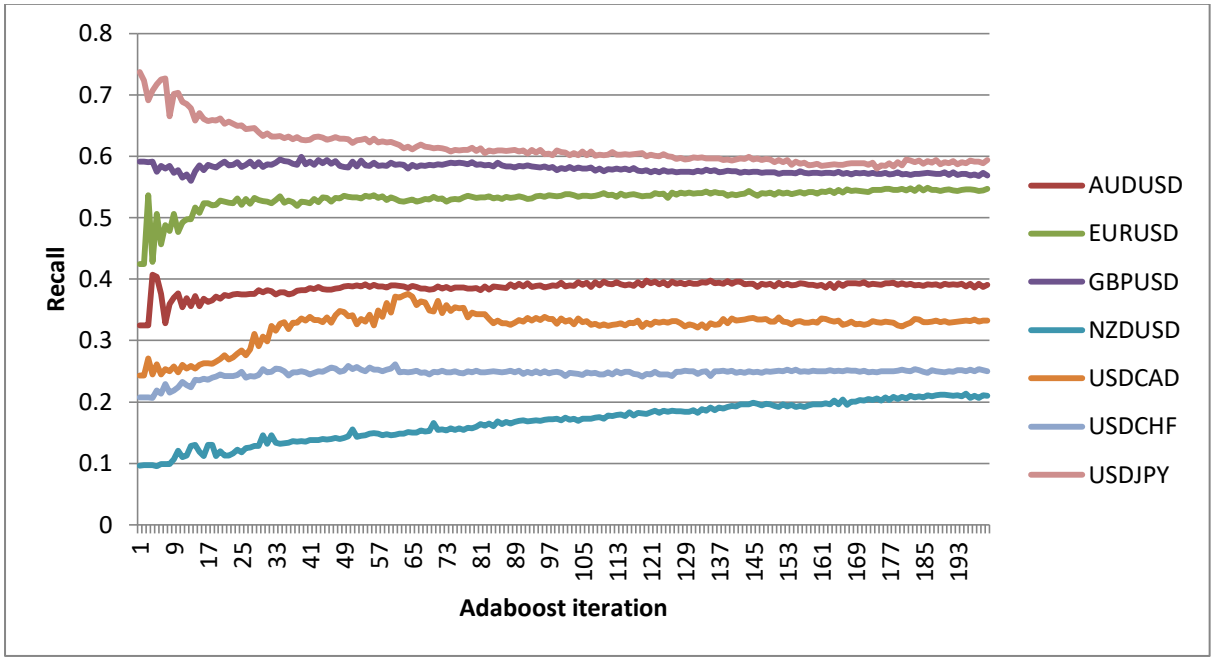


Figure 125: Recall of constructing classifiers in bootstrap with feature-bagging on validation data

Figure 126 shows the precision results of bootstrap and feature-bagging classifiers using outsample data, which are similar to those of the bootstrap classifier (Figure 120). The bootstrap and feature-bagging classifiers using AUDUSD, GBPUSD and USDCAD data perform slightly better than the bootstrap classifiers. The bootstrap and feature-bagging classifier using NZDUSD data does not converge within 200 iterations, whilst the bootstrap classifier using NZDUSD data converges to a precision value of 0.4 by iteration 130. The bootstrap and feature-bagging classifier using USDCHF data converge to a higher precision value of 0.4 than the 0.3 of the corresponding bootstrap classifier.

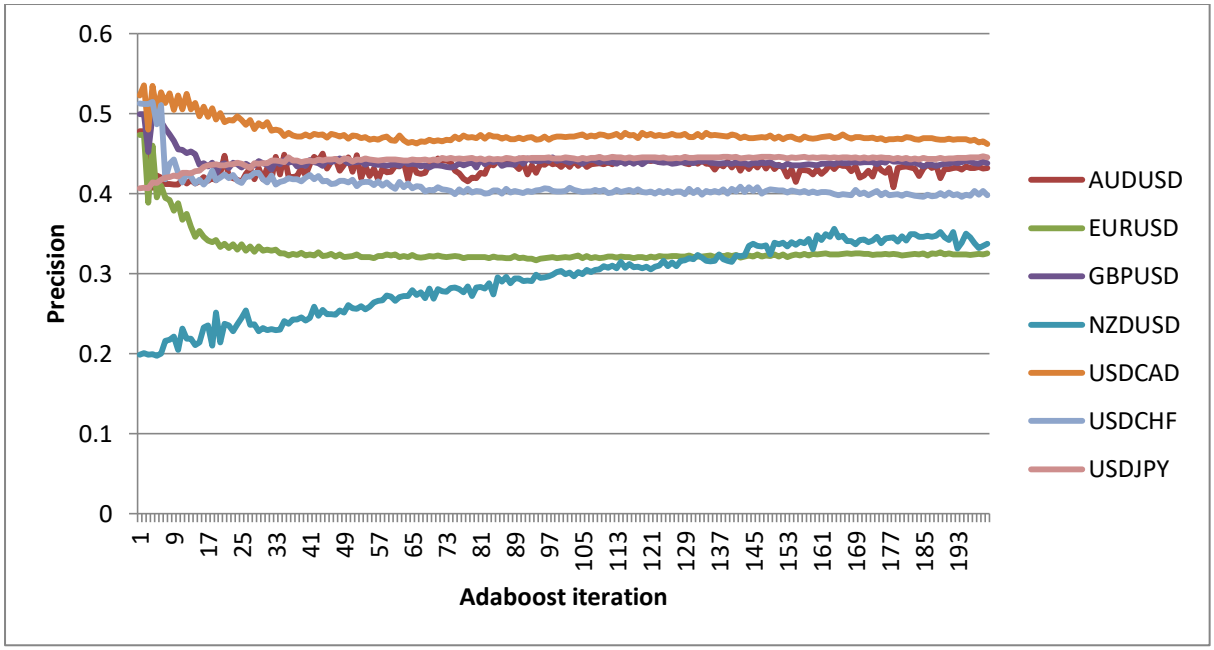


Figure 126: Precision of constructing classifiers in bootstrap with feature-bagging on outsample data

Figure 127 shows the recall values of bootstrap and feature-bagging classifiers using outsample data. The recall values at iteration 200 of classifiers using AUDUSD, EURUSD, NZDUSD, USDCHF, USDJPY data are slightly lower than those of the bootstrap classifiers. The recall values are slightly higher in classifiers using USDCAD data compared to the recall values of the corresponding bootstrap classifier.

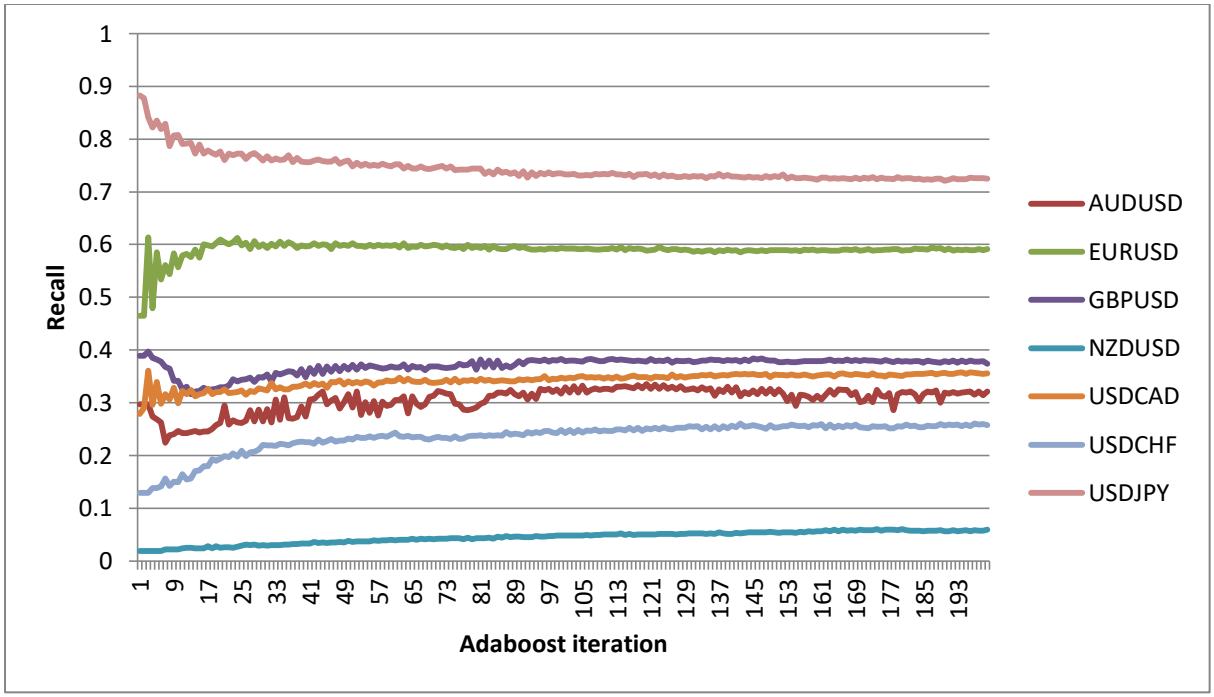


Figure 127: Recall of constructing classifiers in bootstrap with feature-bagging on outsample data

No clear optimal iteration can be determined from the precision or recall results of bootstrap and feature-bagging classifiers using validation data. Figure 128 shows the average precision values of bootstrap and feature-bagging classifiers. The values quickly converge in bootstrap and feature-bagging classifiers using outsample data. However, the average precision values using validation data oscillate slightly. Around the 150th iteration of the Adaboost algorithm, the average precision drops due to the bootstrap and feature-bagging classifier using USDCAD data overfitting, and the average precision fluctuates downwards three times before recovering. The average precision results of bootstrap classifiers, Figure 122, shows overfitting that did not recover, around the 45th iteration. The feature-bagging technique appears to have helped the classification system to reduce overfitting, as intended.

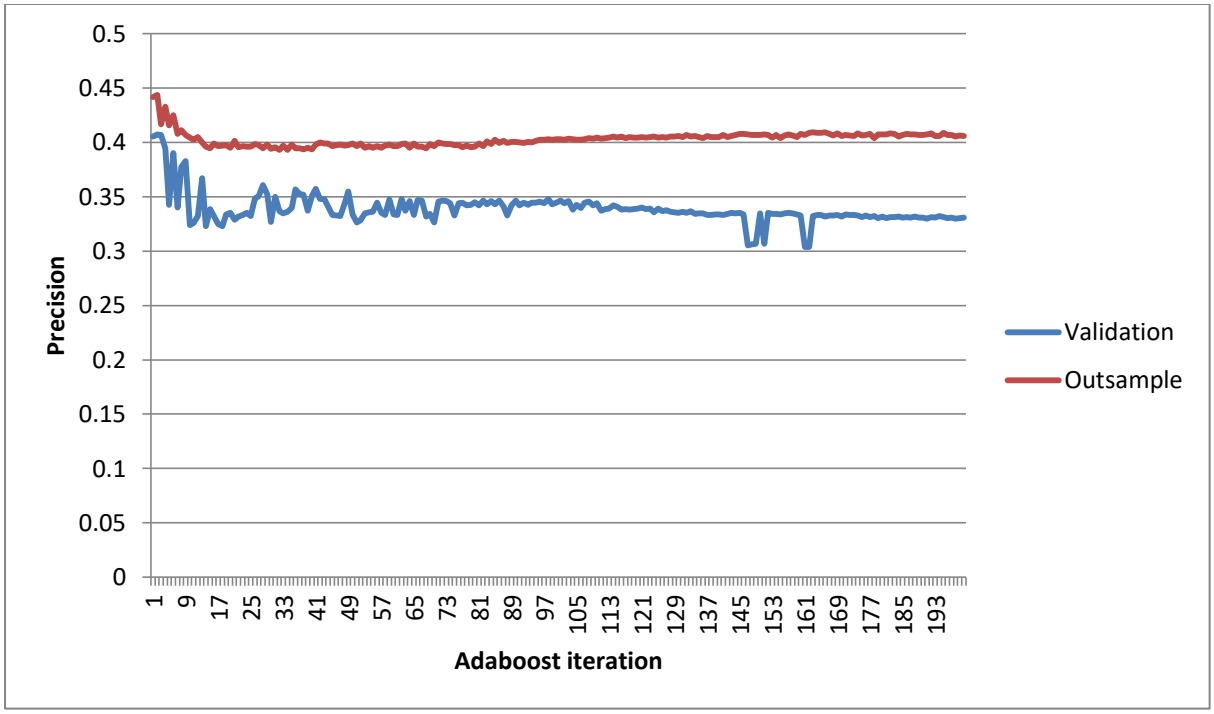


Figure 128: Average precision of constructing bootstrap and feature-bagging classifiers on validation data and outsample data

Figure 129 shows the average recall values of bootstrap and feature-bagging classifiers using validation and outsample data. The values behave similarly to those of the bootstrap classifiers. The recall values of the bootstrap and feature-bagging classifiers converge within 60 iterations.

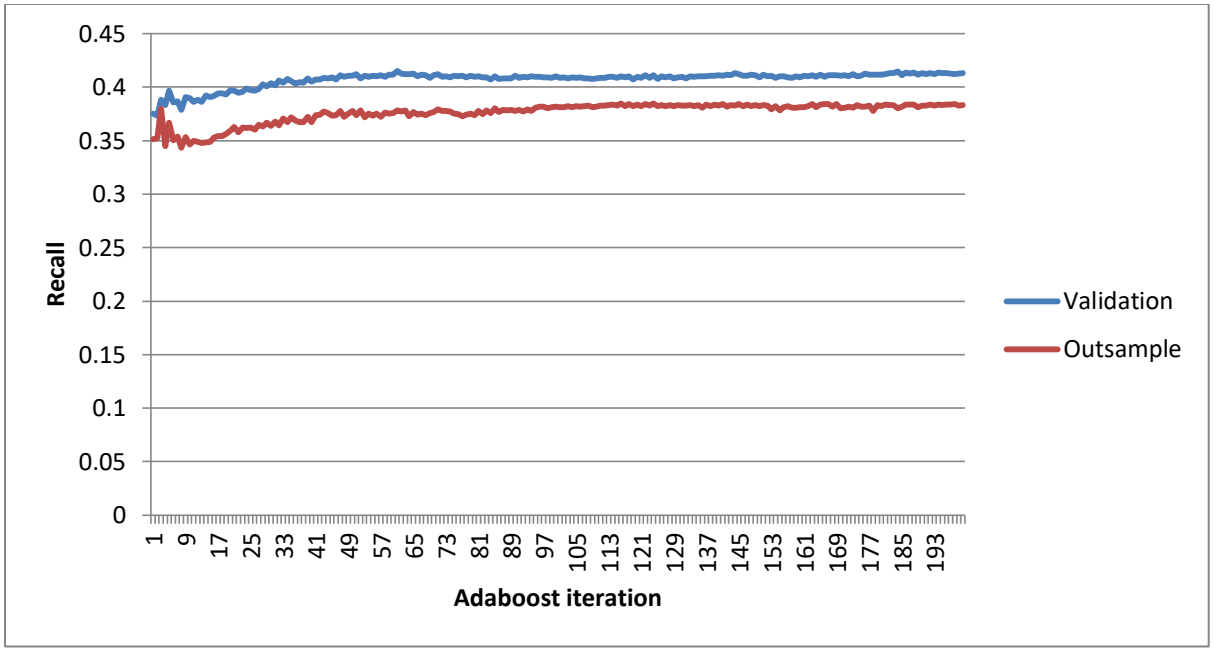


Figure 129: Average recall of constructing bootstrap and feature-bagging classifiers on validation data and outsample data

8.3.3.5 Summary of Precision and Recall Analyses

The average precision of the bootstrap and feature-bagging classifiers at iteration 200 is 0.33 (validation) and 0.41 (outsample), for the bootstrap classifiers is 0.28 (validation) and 0.38 (outsample), and for the individual classifiers is 0.26 (validation) and 0.4 (outsample). The precision of bootstrap and feature-bagging classifiers is higher than those of the other classification systems. Correspondingly, the average recall value of the bootstrap and feature-bagging classifiers at iteration 200 is 0.41 and 0.38, for the bootstrap classifiers is 0.43 and 0.4, and for the individual classifiers is 0.46 and 0.39 from validation and outsample data respectively. The individual classifiers achieved a better recall value using validation data.

The bootstrap and feature-bagging classifiers avoided most of the overfitting that occurred in the individual Adaboost classifier and the bootstrap classifier precision and recall results.

8.3.4 Negative Predictive Value and Specificity Performance

This section will report the negative predictive value and specificity of classification systems at each iteration of the Adaboost algorithm during training. Then report validation and

outsample performance for the individual classifier, the bootstrap aggregation classification system, and bootstrap aggregation with feature bagging classification system.

8.3.4.1 Training Dataset Results

Figures 130, 132 and 134 show the negative predictive values, and Figures 131, 133 and 135 show the specificity of the constructing classifiers at each iteration of the Adaboost algorithm during the training dataset. The training dataset is used to create the classifier.

Figures 130 and 131 show the results of individual classifiers and, as expected, the negative predictive value and specificity values generally increase with each iteration of the Adaboost algorithm. An abnormal change in the negative predictive value and specificity values is observed at the 16th iteration of the Adaboost algorithm (as can be observed in the precision and recall values in this phase, Figures 106 and 107, but for accuracy, Figure 85). This suggests that the Adaboost algorithm was forced into a more predictive state (in accuracy) resulting in learning highly weighted data points that affected the negative predictive value.

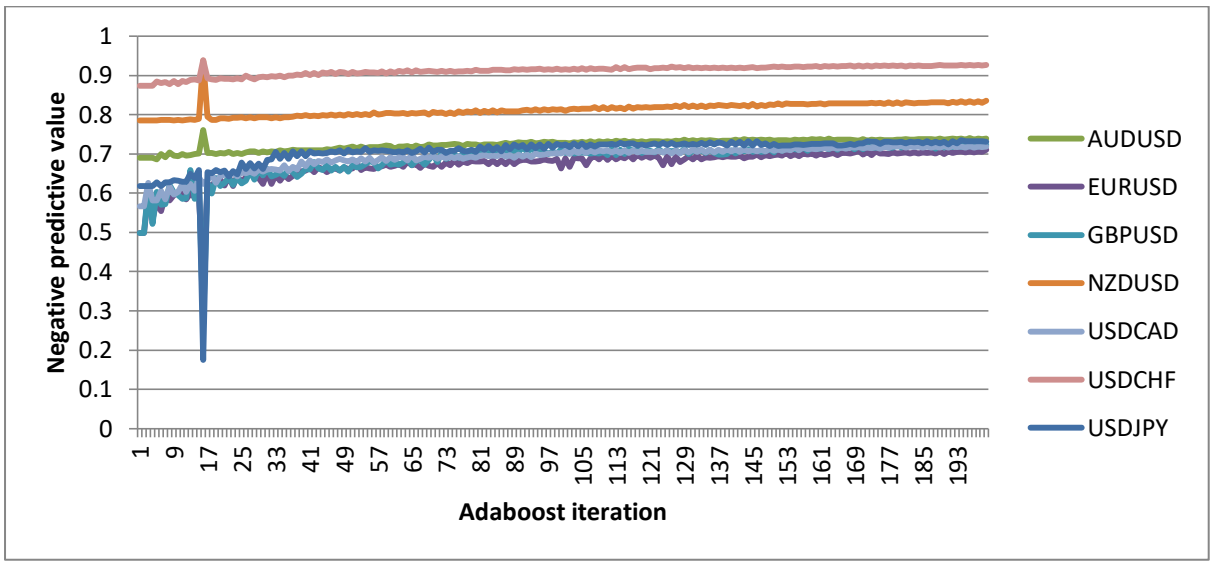


Figure 130: Negative predictive value of constructing classifiers during the training phase

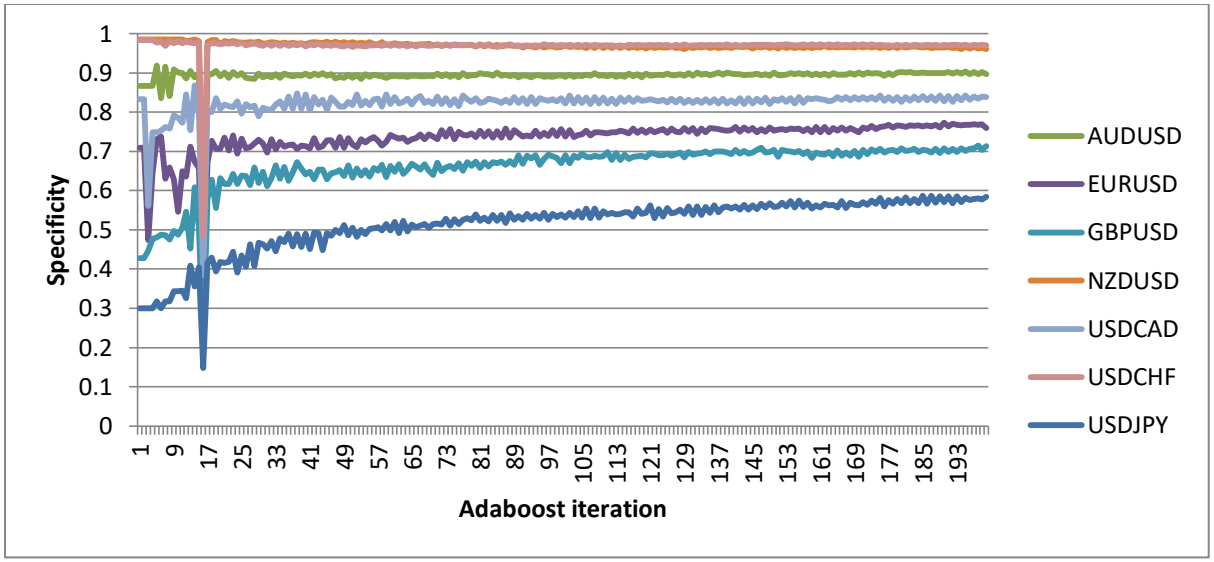


Figure 131: Specificity of constructing classifiers during the training phase

Figures 132 and 133 show the negative predictive value and specificity results for the bootstrap classifiers at each iteration of the Adaboost algorithm. As expected, the values generally increase as Adaboost iterates; the results are smoother than for the individual classifiers, but the abnormality at the 16th iteration is also present.

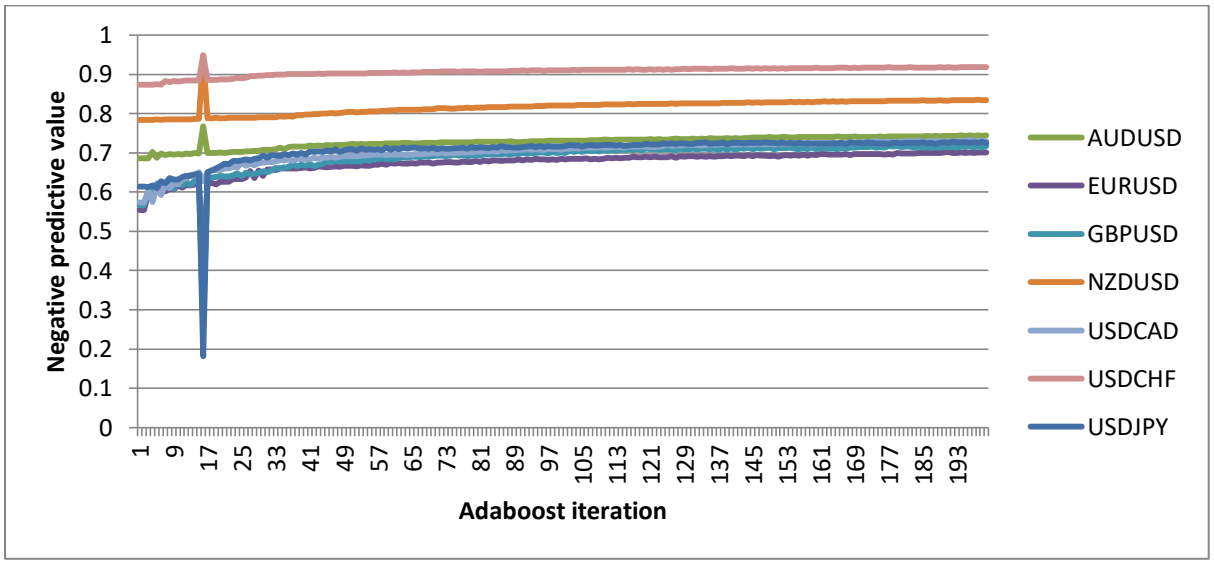


Figure 132: Negative predictive value of constructing classifiers using the bootstrap aggregation technique during the training phase

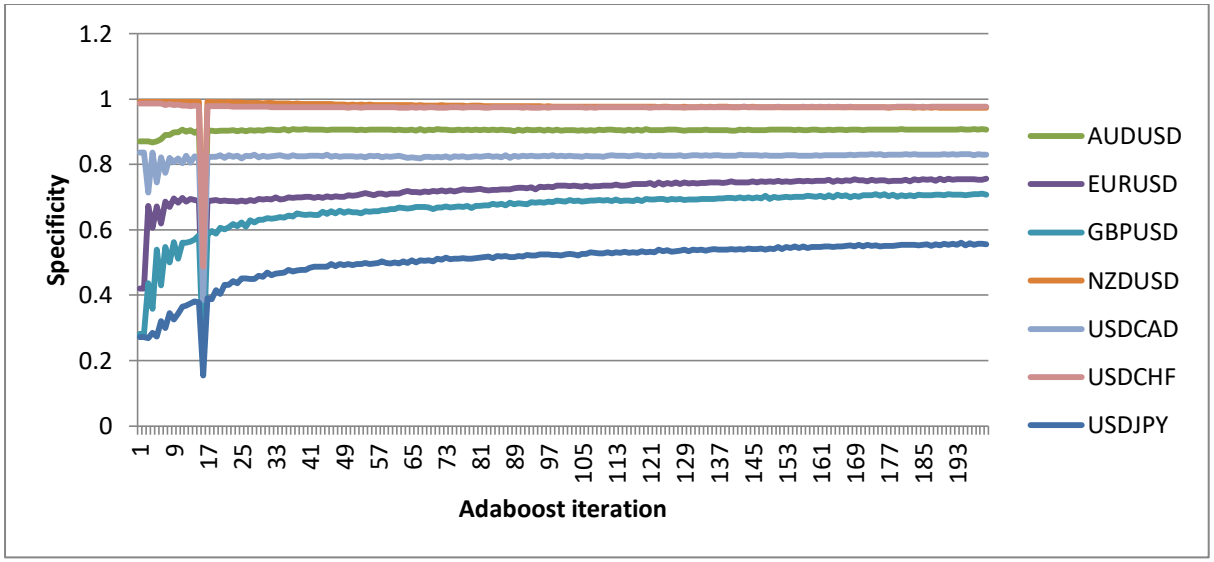


Figure 133: Specificity of constructing classifiers using the bootstrap aggregation technique during the training phase

Figures 134 and 135 show the negative predictive value and specificity of the bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm. These are also smoother than for the individual classifiers, and the abnormality at the 16th iteration is also present.

As with the previous measures, the behaviour in negative predictive value and specificity support the view that the classifiers are learning the training data.

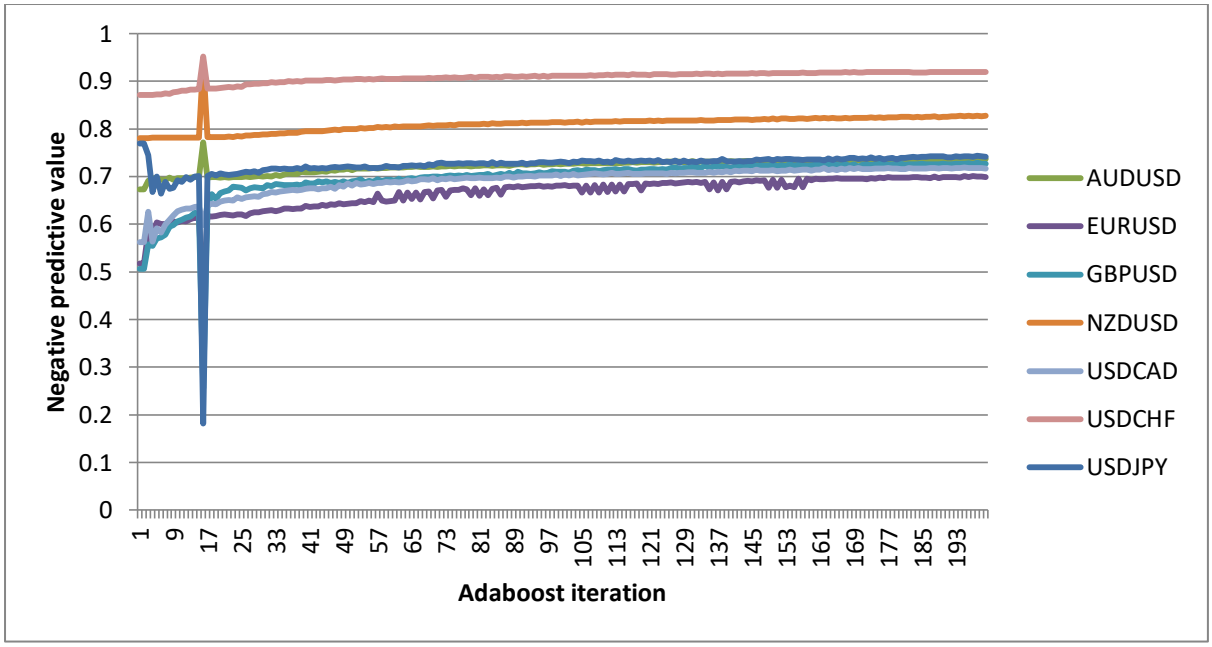


Figure 134: Negative predictive value of constructing classifiers in bootstrap with feature-bagging during the training phase

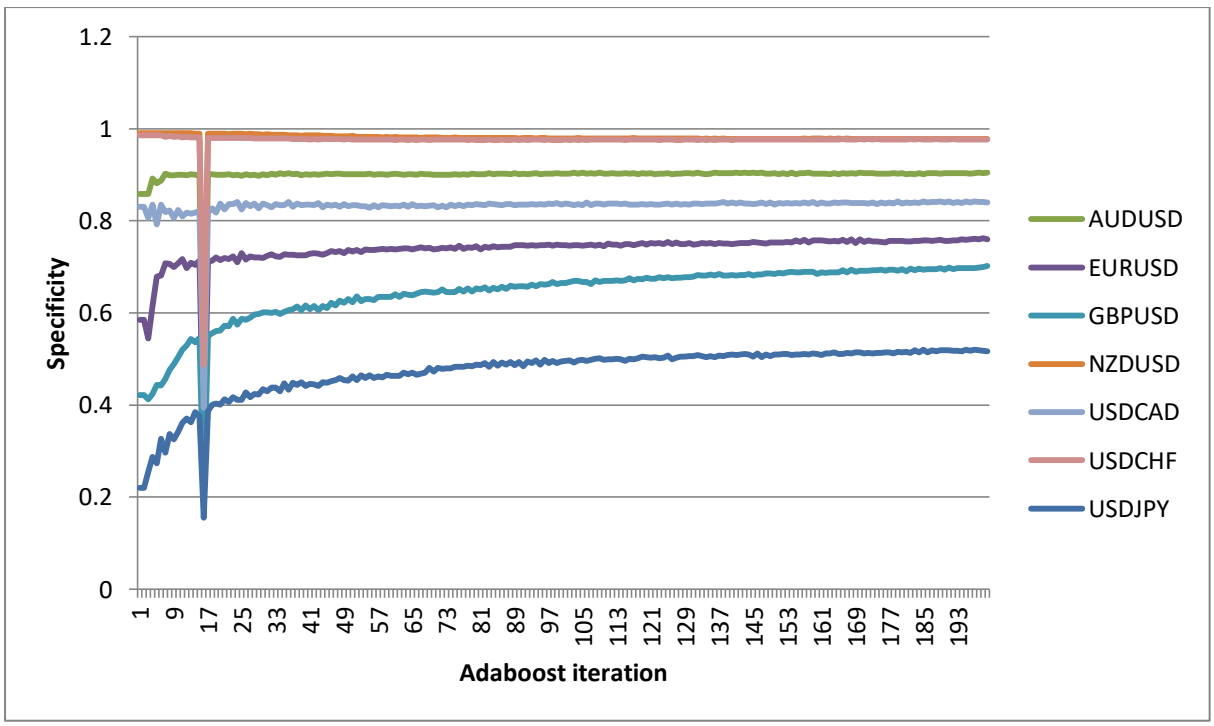


Figure 135: Specificity of constructing classifiers in bootstrap with feature-bagging during the training phase

8.3.4.2 Individual Classifier

Figure 136 shows the negative predictive values of individual classifiers at each iteration of the Adaboost algorithm using validation data. The precision jumps previously observed in individual classifiers using AUDUSD and USDCAD data (Figure 112) are also observed in the negative predictive value results. The negative predictive values of individual classifiers using EURUSD and NZDUSD data increase slightly with iteration, while those using USDCHF data did not change. For the classifiers using GBPUSD and USDJPY, a gradual increase is observed. The negative predictive value of the individual classifiers at iteration 200 is between 0.67 and 0.83, except for the individual classifier using USDCAD data which achieves only random performance.

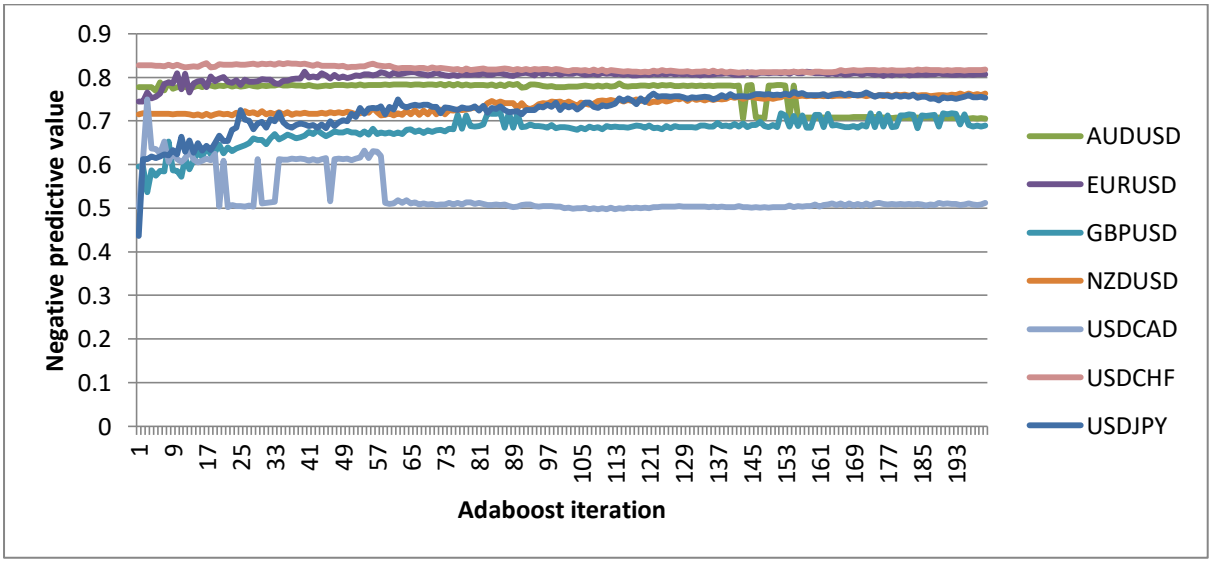


Figure 136: Negative predictive value of constructing individual classifiers on validation data

Figure 137 shows the specificity of individual classifiers at each iteration of the Adaboost algorithm using validation data. Large drops in specificity value in individual classifiers using AUDUSD and USDCAD data are observed, whereas no drops in recall value were previously observed, Figure 113. However, the overall accuracy results of individual classifiers using AUDUSD and USDCAD data (shown in Figure 91) did show these large drops. The specificity of classifiers using EURUSD and USDJPY generally increase with iteration, but for EURUSD data there is a decrease around the 70^{th} iteration. The specificity of classifiers using NZDUSD and USDCHF data generally decrease with iteration. The specificity of the classifier using GBPUSD data rises slightly before converging. The specificity of individual classifiers at

iteration 200 is mixed; for the NZDUSD and USDCHF the value is 0.71 and 0.76 respectively, but for AUDUSD, EURUSD and USDCAD performance is random and classifiers using GBPUSD and USDJPY data only achieved 0.25 and 0.35 specificity respectively. The drops in specificity value in the individual classifiers using AUDUSD and USDCAD are a sign of overfitting; had overfitting not occurred, a value above 0.65 might have been expected.

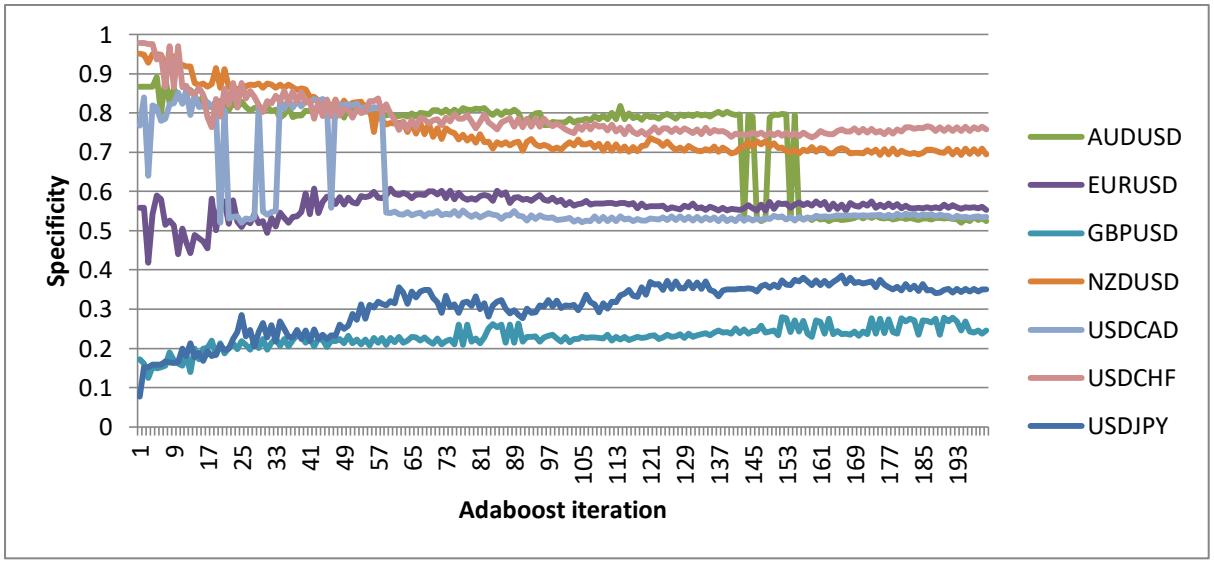


Figure 137: Specificity of constructing individual classifiers on validation data

Figure 138 shows the negative predictive value of individual classifiers at each iteration of the Adaboost algorithm using outsample data. Except for the individual classifier using USDJPY data, which increases before converging within 50 iterations, the values of individual classifiers remain relatively steady.

The results in the outsample dataset are similar to the results in the validation dataset, in that all classifiers converge rather quickly. However, the drops in negative predictive value observed in the validation data are not observed in the outsample dataset. The negative predictive values of the individual classifiers are 0.45, 0.65, 0.65, 0.69, 0.72, 0.8 and 0.85 on outsample data. Only the classifier using NZDUSD data is poor at classifying unprofitable technical analysis interpretations; the other classifiers are considerably better than random at identifying unprofitable technical analysis interpretations.

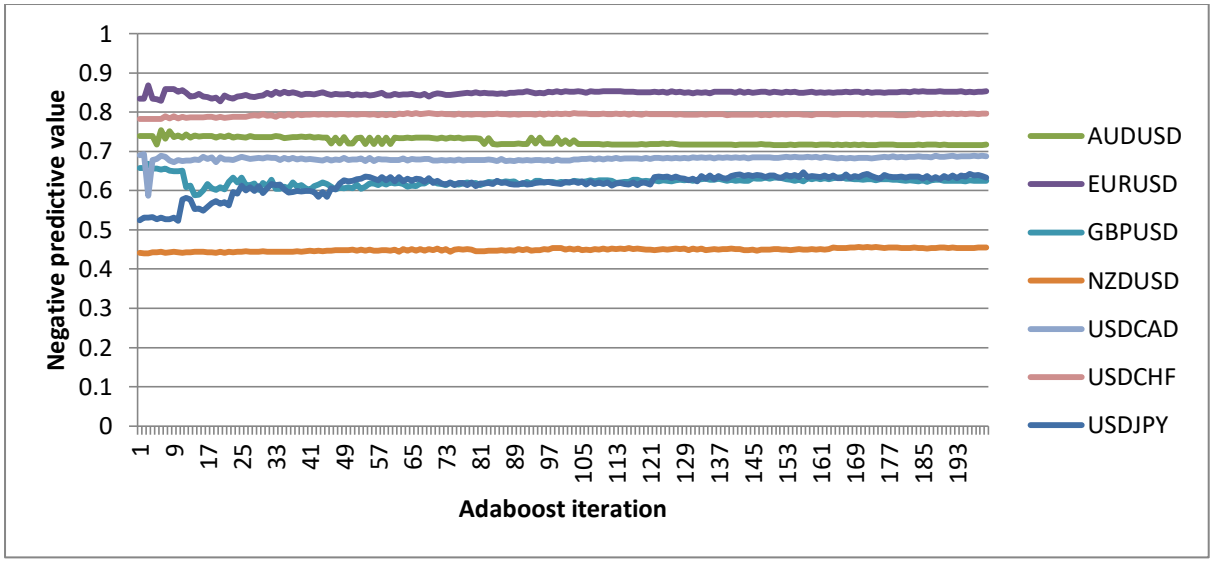


Figure 138: Negative predictive value of constructing individual classifiers on outsample data

Figure 139 shows the specificity of individual classifiers at each iteration of the Adaboost algorithm using outsample data. Except for the worst performing individual classifier that used UDSJPY data, for which the specificity gradually increased to 0.32, the specificity of all the other individual classifiers converged quickly. At the 200th iteration, two classifiers achieved 0.65 specificity and four classifiers achieved around 0.82. Comparing these results with the corresponding recall results (Figure 115), individual classifiers are better at identifying unprofitable technical analysis interpretations than at identifying profitable interpretations.

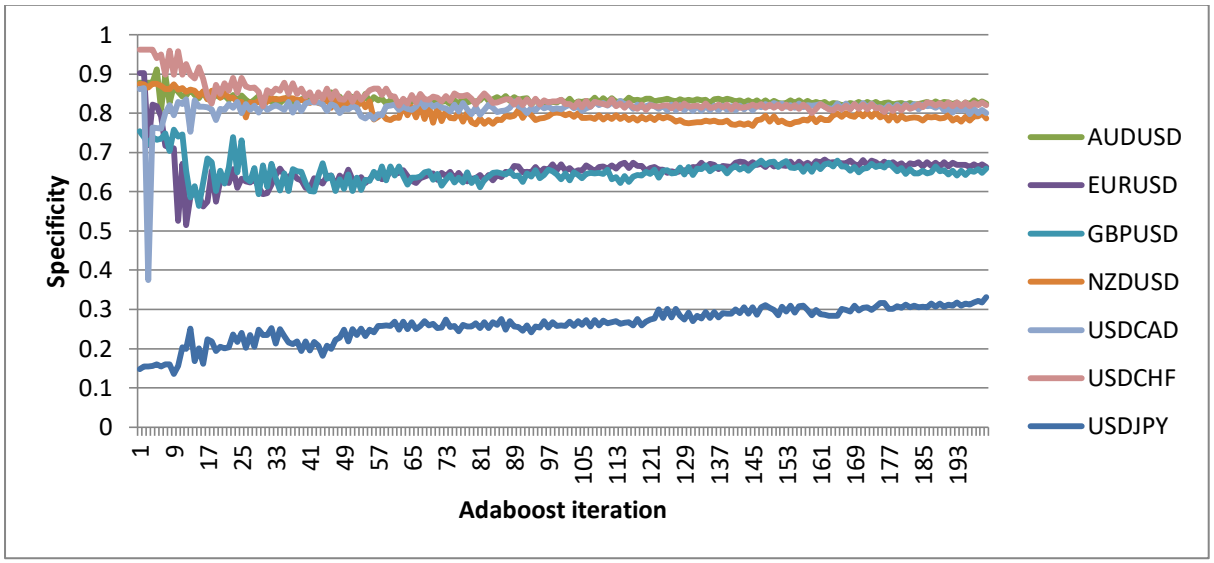


Figure 139: Specificity of constructing individual classifiers on outsample data

Figure 140 shows the average negative predictive value of individual classifiers on validation and outsample data. The individual classifiers achieved better than random performance, with values of 0.72 and 0.68 for validation and outsample data respectively. Comparing these results with the average precision results of 0.26 and 0.40 in Figure 116, individual classifiers are better at classifying unprofitable technical analysis interpretations as unprofitable, and worse than random at classifying profitable technical analysis interpretations as profitable.

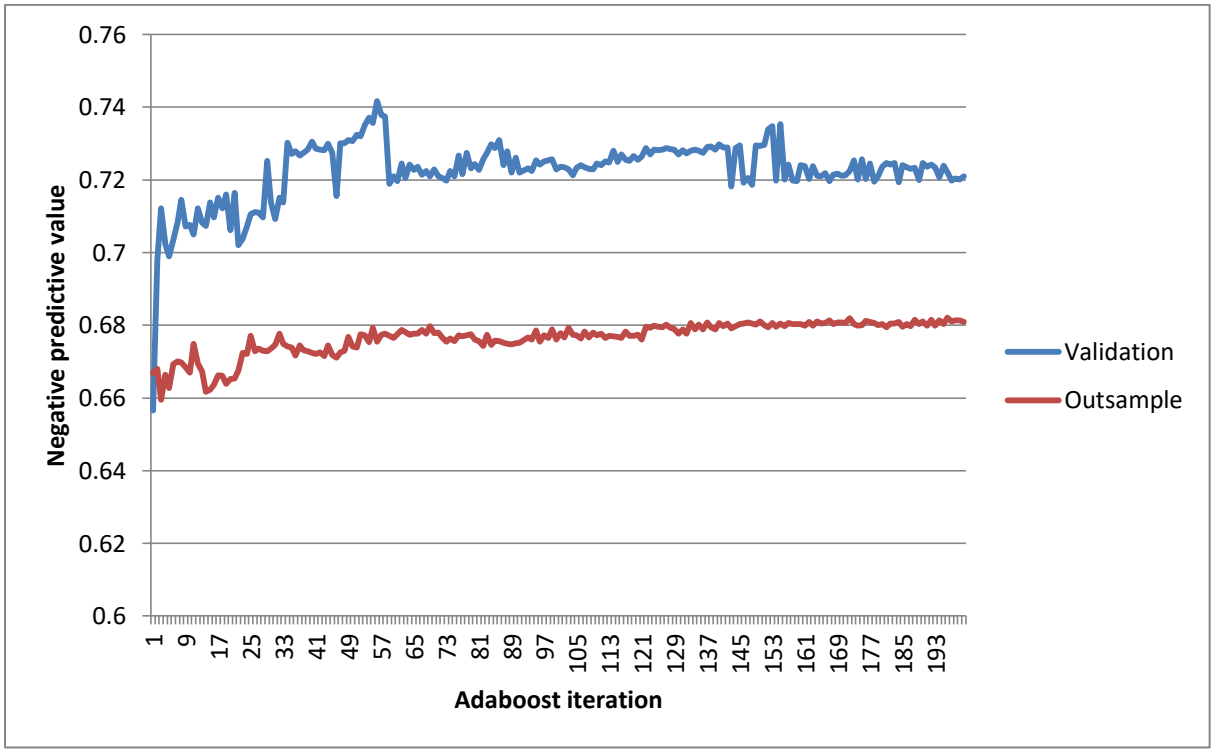


Figure 140: Average negative predictive value of constructing individual classifiers on validation and outsample data

Figure 141 shows the average specificity of individual classifiers on validation and outsample data. The average specificity of individual classifiers at the 200th iteration of the Adaboost algorithm is 0.52 and 0.7 for validation and outsample data respectively. The drops in specificity of individual classifiers using AUDUSD and USDCAD validation data observed in Figure 137 are also produced in the average specificity results. Choosing individual classifiers before the specificity drops would produce an average specificity of 0.62 on validation data.

Comparing the average specificity results with the average recall results of 0.46 and 0.39 in Figure 117, it can again be seen that individual classifiers are better at identifying the

unprofitable technical analysis interpretations, and are worse than random at identifying the profitable technical analysis interpretations.

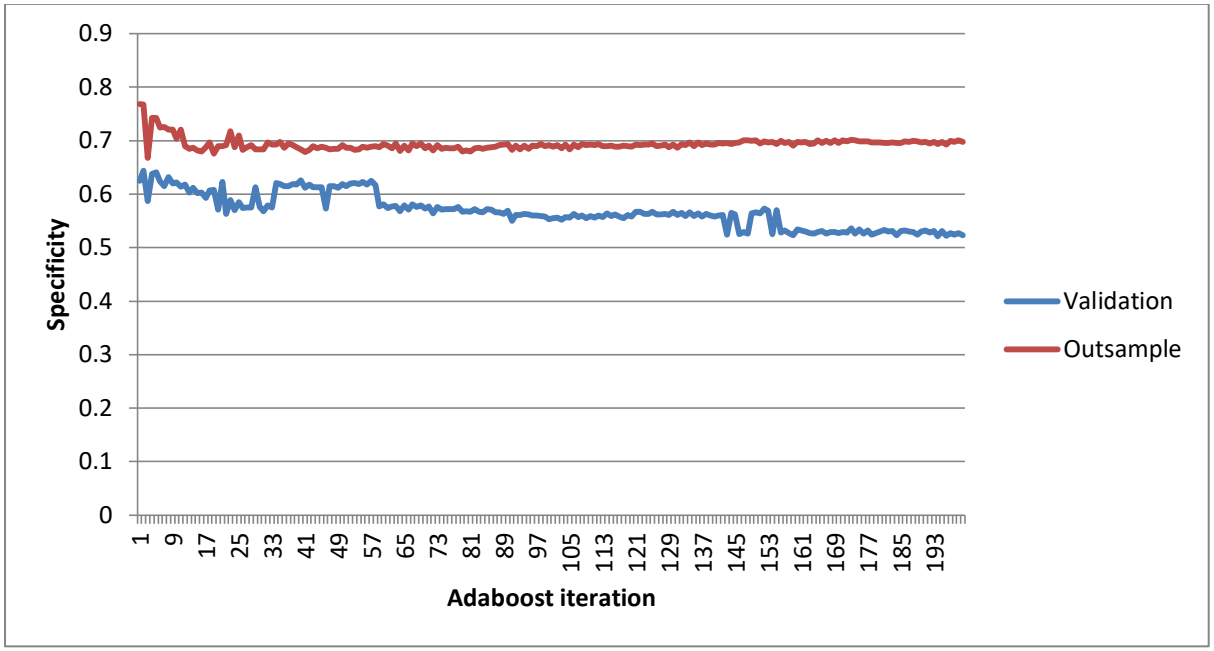


Figure 141: Average specificity of constructing individual classifiers on validation and outsample data

8.3.4.3 Bootstrap Aggregation Classifier

Figures 142 and 143 shows the negative predictive value and specificity of bootstrap classifiers at each iteration of the Adaboost algorithm using validation data. The negative predictive values again converge quickly, with the classifier using USDJPY data being slower, taking about 80 iterations. A drop in negative predictive value from 0.62 to 0.52 is observed in the bootstrap classifier using USDCAD data, with the other classifiers achieving a value of at least 0.7. The corresponding results for individual classifiers (Figure 136) show that the individual classifiers using AUDUSD and USDCAD data have drops in this parameter. The value dropped in the bootstrap classification system only for USDCAD data. The bootstrap classifier using AUDUSD avoided the overfitting that occurred in the individual classifier; this is clear also in the specificity results. The results are similar to those for the individual classifier (Figure 137). The specificity values are smoother than in the individual classifier results.

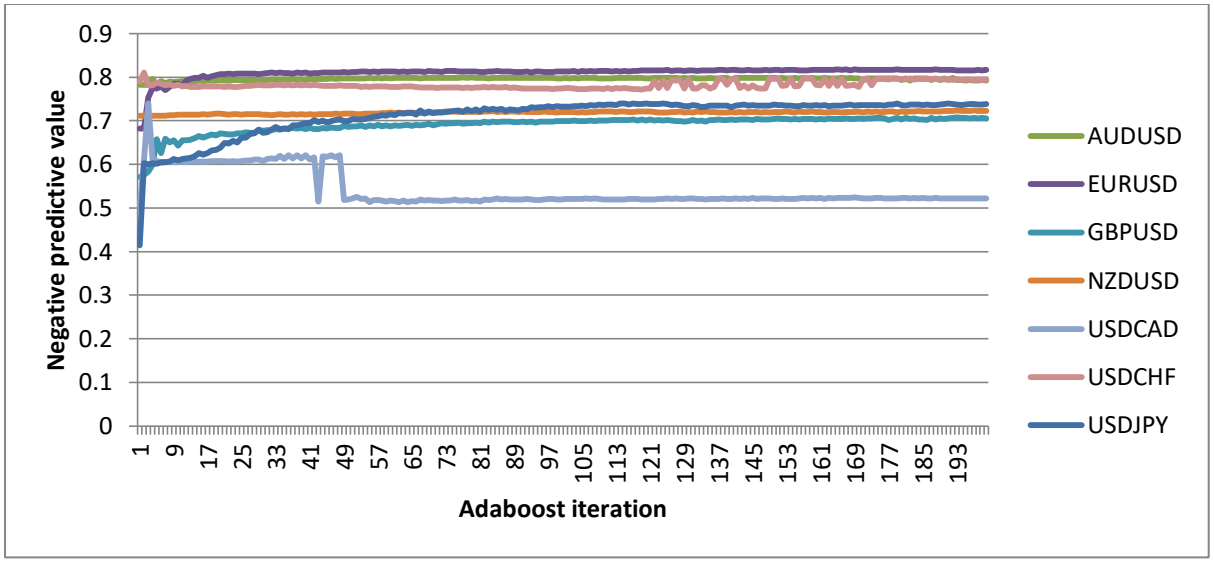


Figure 142: Negative predictive value of constructing classifiers using bootstrap on validation data

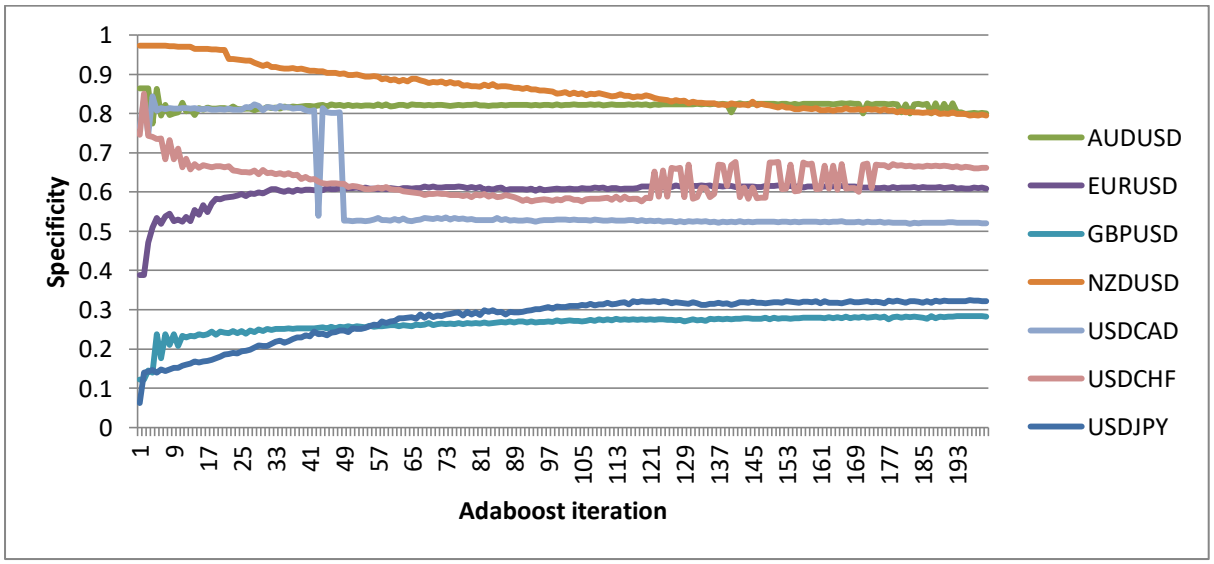


Figure 143: Specificity of constructing classifiers using bootstrap on validation data

Figures 144 and 145 show the negative predictive value and specificity of bootstrap classifiers at each iteration of the Adaboost algorithm on outsample data. The results are smoother and follow closely those observed for individual classifiers in Figures 138 and 139.

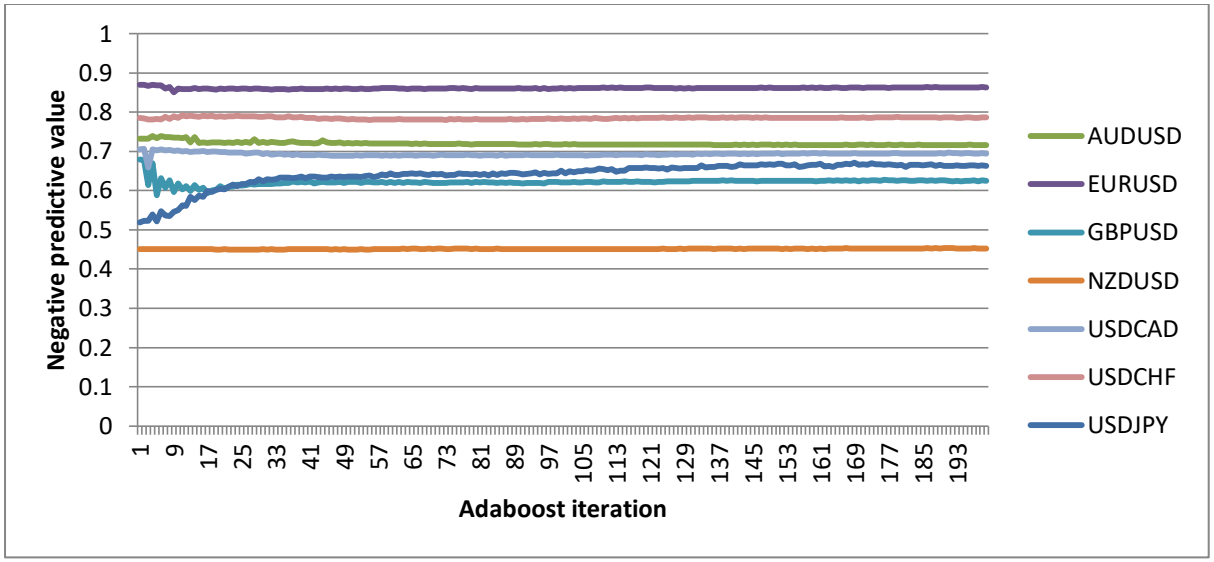


Figure 144: Negative predictive value of constructing classifiers using bootstrap on outsample data

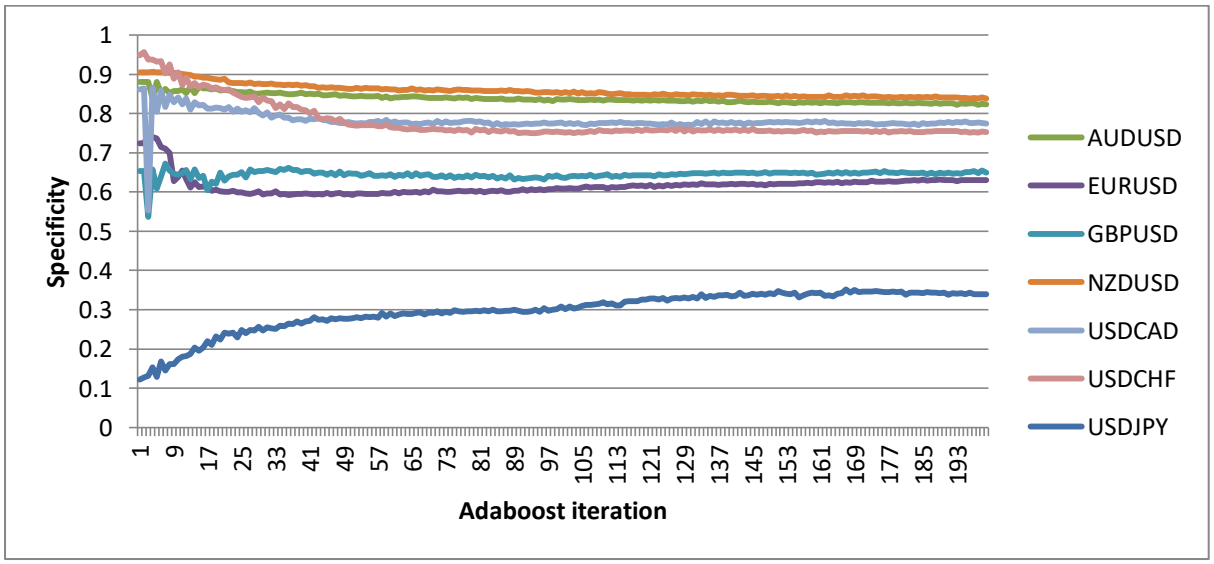


Figure 145: Specificity of constructing classifiers using bootstrap on outsample data

Figure 146 shows the average negative predictive values of bootstrap classifiers at each iteration of the Adaboost algorithm. Comparing the results with those of the individual classifiers (Figure 140), the average negative predictive values are smoother with only one drop (rather than two) observed, at the 48th iteration. The bootstrap aggregation technique reduced overfitting and slightly increased performance in classifying unprofitable technical analysis interpretations; the average negative predictive value rises from 0.72 to 0.73 on validation data and rises from 0.68 to 0.69 on outsample data.

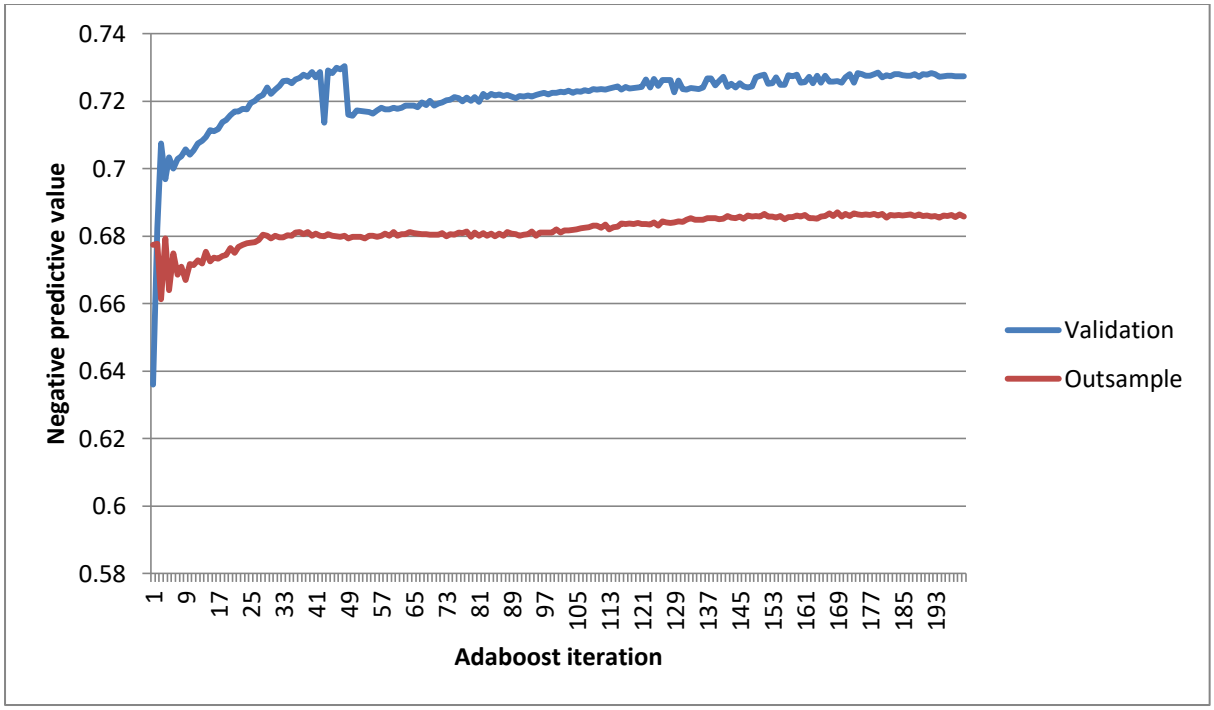


Figure 146: Average negative predictive value of constructing bootstrap classifiers on validation data and outsample data

Figure 147 shows the average specificity of bootstrap classifiers at each iteration of the Adaboost algorithm. Similar to the negative predictive value results, the classifiers using the bootstrap aggregation technique produce smoother results compared to the individual classifier results (Figure 141), and again only one drop in specificity performance on validation data is observed, rather than two.

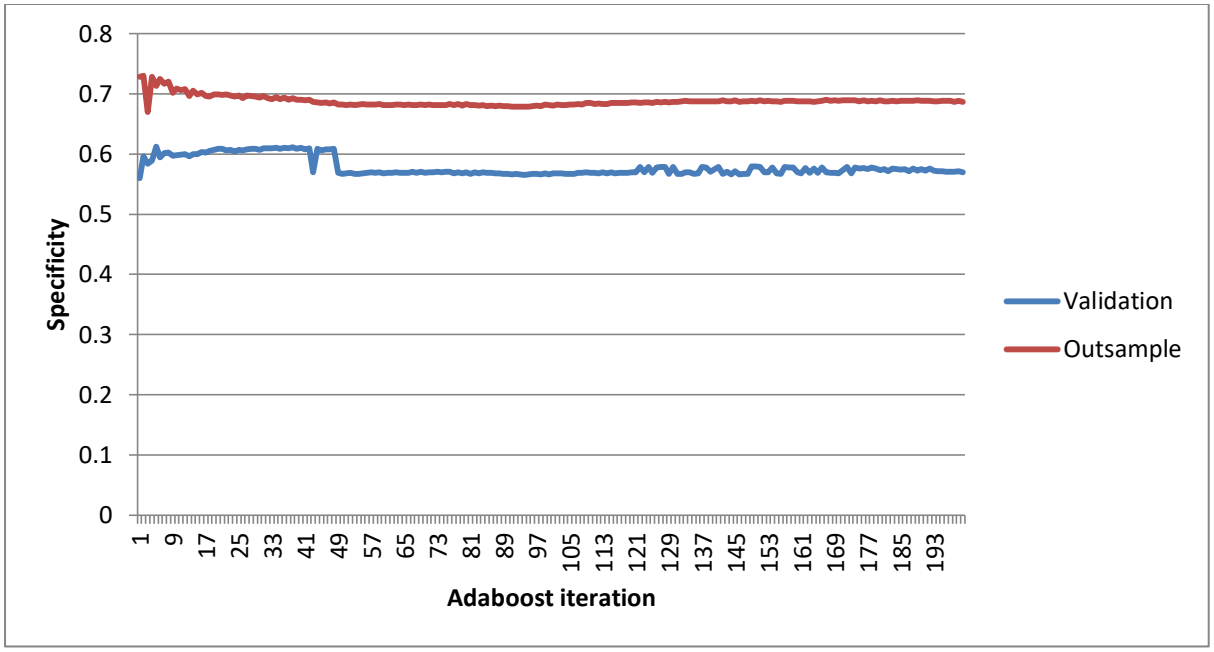


Figure 147: Average specificity of constructing bootstrap classifiers on validation data and outsample data

8.3.4.4 Bootstrap Aggregation and Feature-bagging Classifier

Figure 148 shows the negative predictive value of bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm using validation data. The values are similar to those for the bootstrap classifiers, with both classification systems achieving at least a value of 0.7, except for USDCAD data (about 0.5 in both systems).

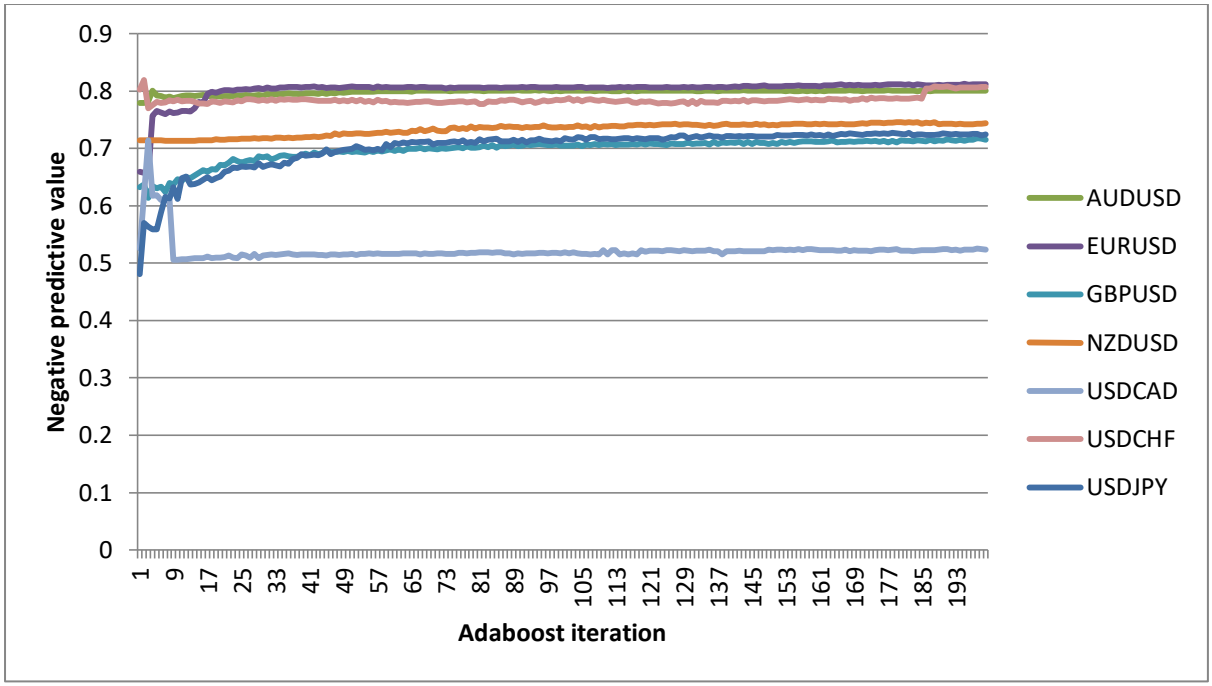


Figure 148: Negative predictive value of constructing classifiers in bootstrap with feature-bagging on validation data

Figure 149 shows the specificity values of bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm using validation data. The specificity results are broadly similar to the specificity results of bootstrap classifiers. However, the classifier using USDCAD data recovered from a specificity drop from 0.8 to 0.51 back up to 0.8 whilst the bootstrap classifier using USDCAD data did not recover from a similar drop. Additionally, the specificity value of the bootstrap and feature-bagging classifier using USDCHF data at the 200^{th} iteration is 0.84 which is much higher than the 0.68 achieved by the of the individual Adaboost classifier (Figure 143).

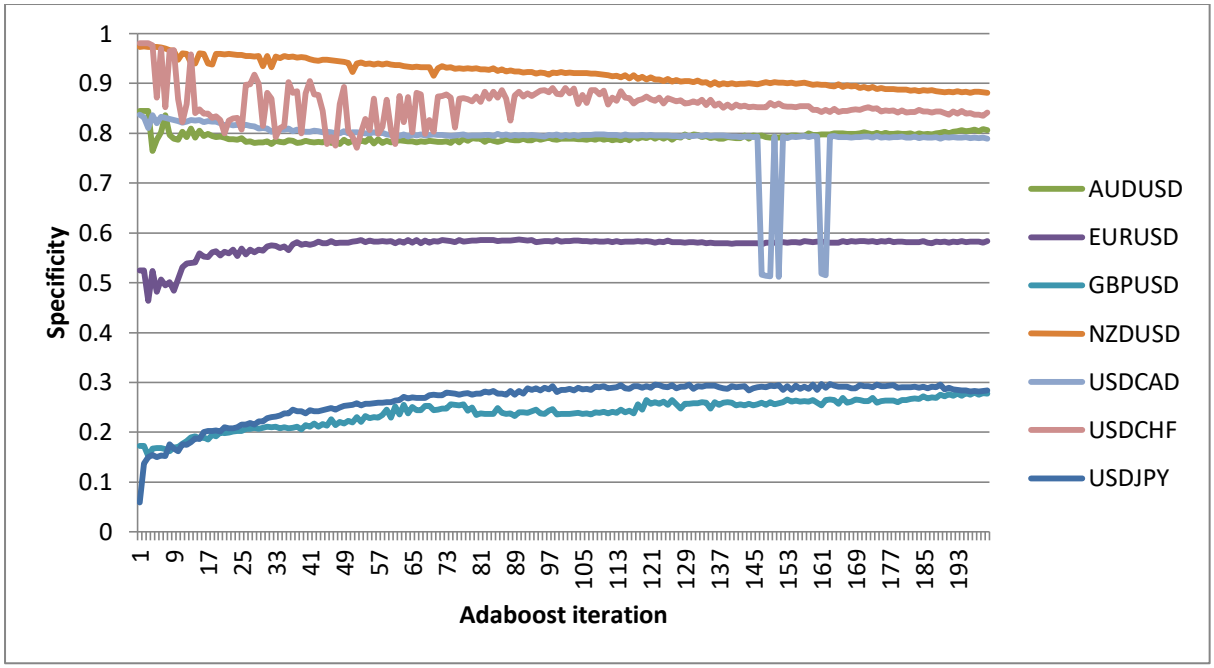


Figure 149: Specificity of constructing classifiers in bootstrap with feature-bagging on validation data

Figures 150 and 151 show the negative predictive value and specificity values of bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm using outsample data. The results closely resembled those of the bootstrap classifiers.

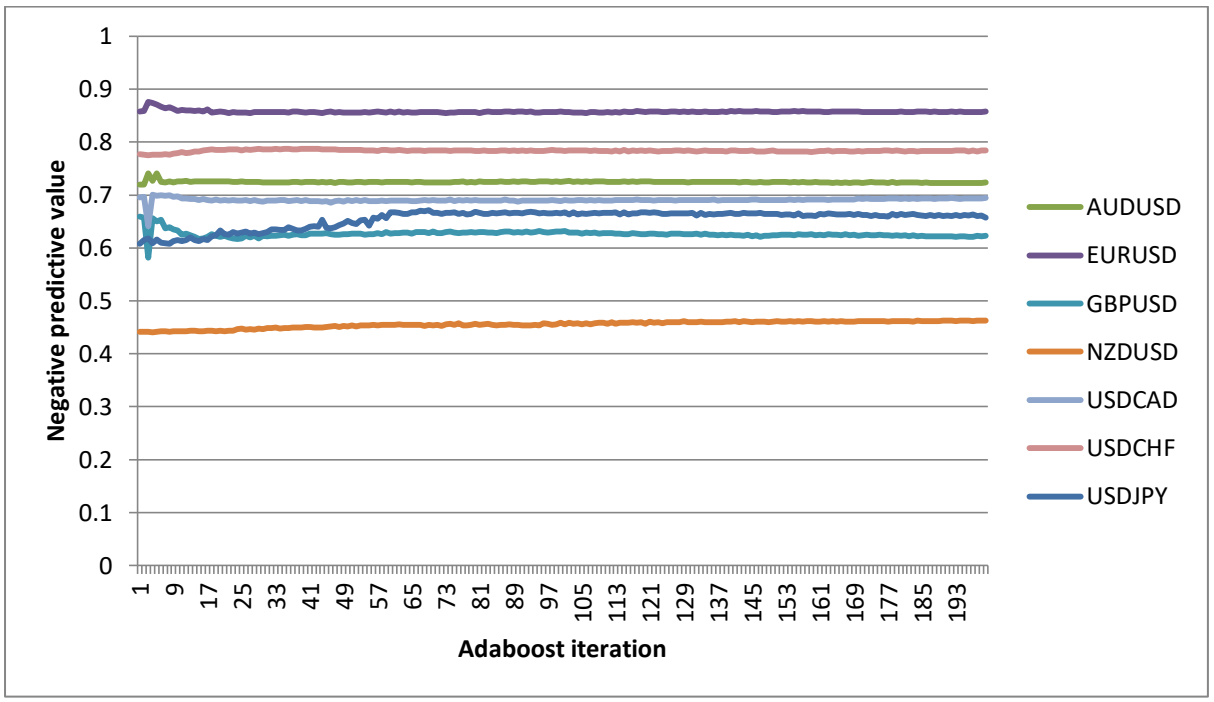


Figure 150: Negative predictive value of constructing classifiers in bootstrap with feature-bagging on outsample data

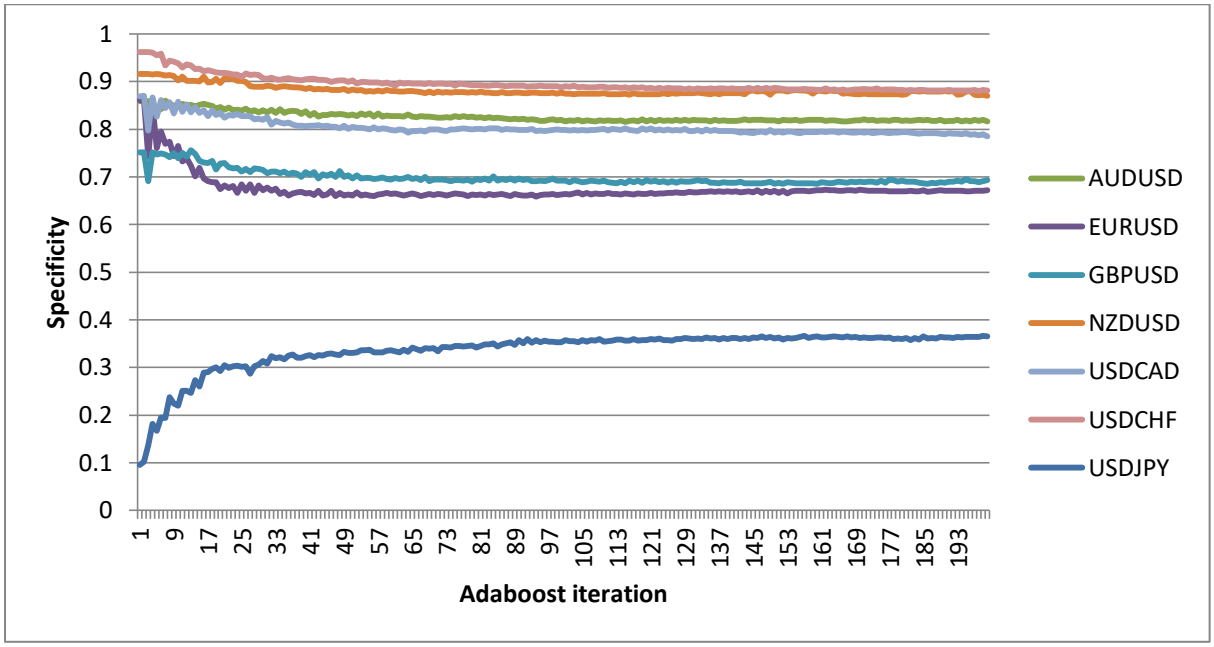


Figure 151: Specificity of constructing classifiers in bootstrap with feature-bagging on outsample data

Figure 152 shows the average negative predictive value of bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm using validation and outsample data. Comparing the results with individual classifier results and the results from bootstrap classifiers, the average negative predictive value at the 200th iteration on outsample data is the same as the bootstrap classifier and is slightly better than the individual classifier. The average negative predictive value results show two instances of overfitting of individual classifiers using validation data and one instance of overfitting of a bootstrap classifier using validation data. None of the bootstrap and feature-bagging classifiers using validation data overfitted. The difference in the average negative predictive value of each classification system at the 200th iteration of the Adaboost algorithm is marginal.

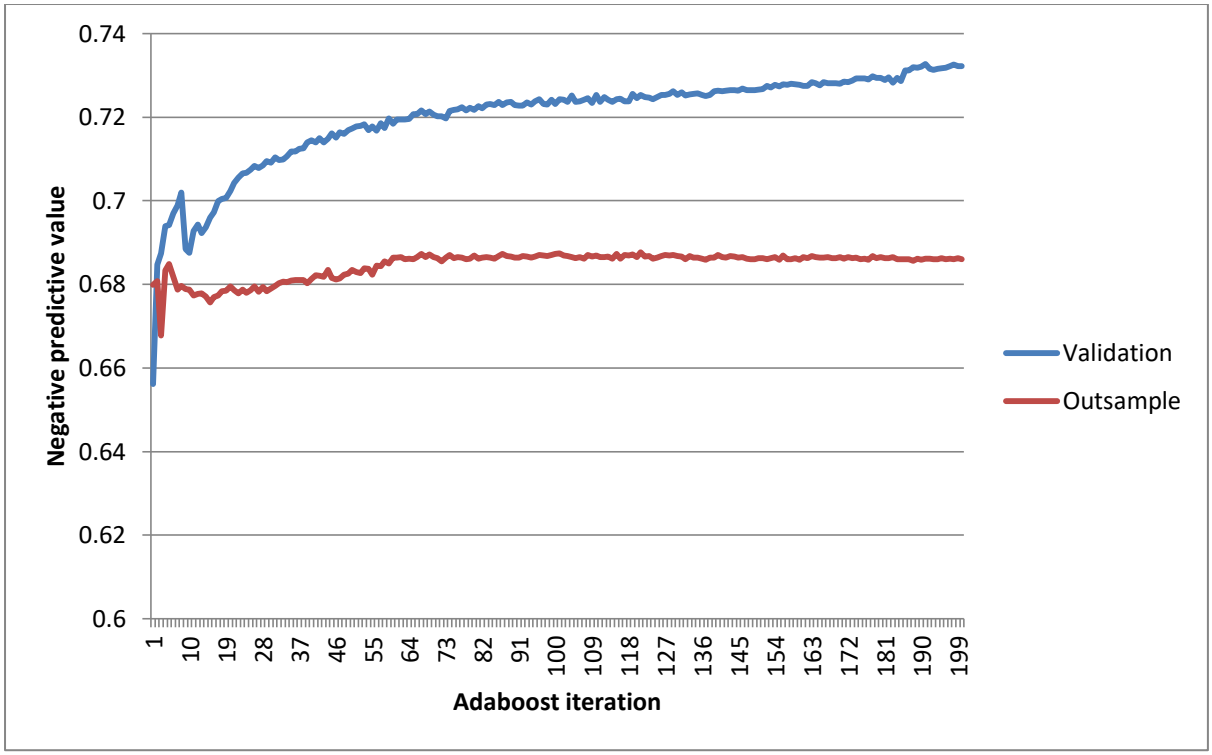


Figure 152: Average negative predictive value of constructing bootstrap and feature-bagging classifiers on validation data and outsample data

Figure 153 shows the average specificity value of bootstrap and feature-bagging classifiers at each iteration of the Adaboost algorithm using validation and outsample data. Comparing these results with individual classifiers, the bootstrap and feature-bagging classifiers performed the same on outsample data but achieved a 0.1 increase in specificity on validation data. Comparing these results with bootstrap classifiers, the bootstrap and feature-bagging classifiers performed slight better on outsample data but achieved a 0.07 increase in specificity on validation data.

The average specificity results show two instances of overfitting of individual classifiers using validation data and one instance of overfitting of bootstrap classifiers using validation data. The classifiers using the bootstrap aggregation technique and feature-bagging did overfit once but recovered.

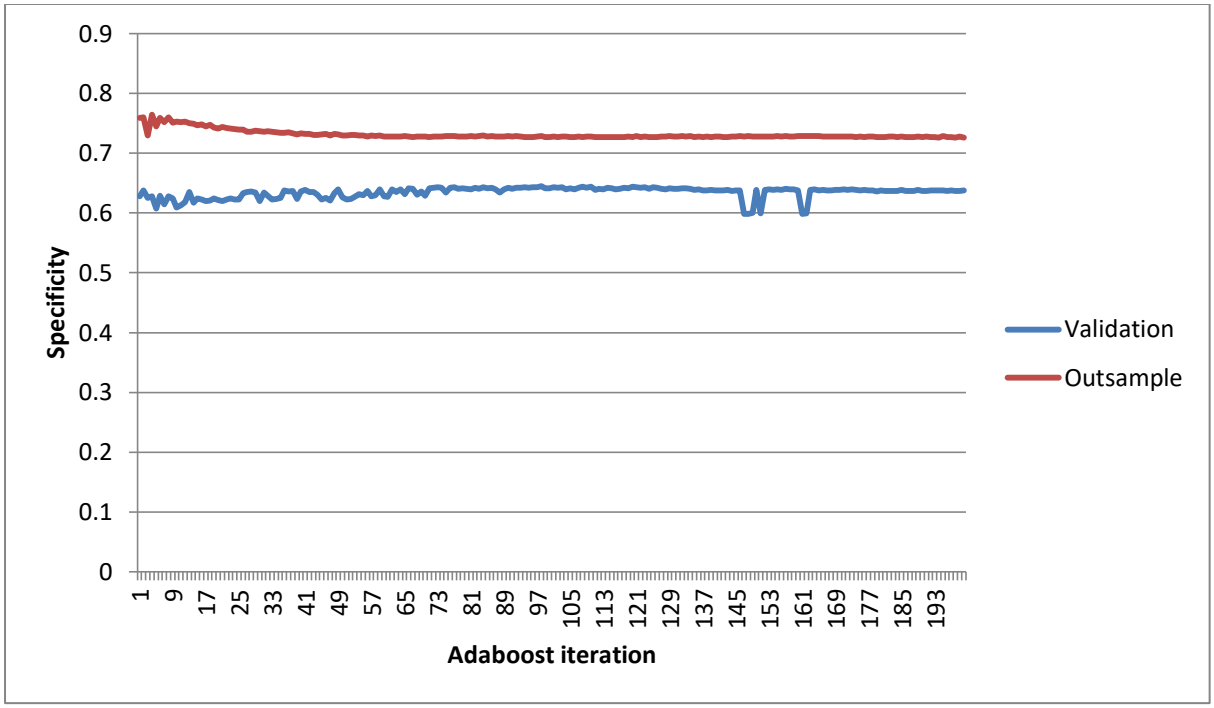


Figure 153: Average specificity of constructing bootstrap and feature-bagging classifiers on validation data and outsample data

8.3.4.5 Summary of Negative Predictive Value and Specificity Analyses

The average negative predictive value of the bootstrap and feature-bagging classifiers at iteration 200 is 0.73 (validation) and 0.69 (outsample), for the bootstrap classifiers is 0.73 (validation) and 0.68 (outsample), and for the individual classifiers is 0.72 (validation) and 0.68 (outsample). The average negative predictive values of these classification systems are very similar. Correspondingly, the average specificity value of the bootstrap and feature-bagging classifiers at iteration 200 is 0.65 and 0.74, for the bootstrap classifiers is 0.58 and 0.7, and for the individual classifiers is 0.53 and 0.7 from validation and outsample data respectively. The bootstrap and feature-bagging classification system achieved better specificity on validation and outsample data.

The bootstrap and feature-bagging classifiers avoided most of the overfitting that occurred in the individual Adaboost classifier and in the bootstrap classifier negative predictive value and specificity results.

8.3.5 F_1 Score results

The F_1 score (harmonic mean) is a metric that is used to assess the performance of a binary classifier, Section 2.5. This section will calculate the F_1 score of the precision and recall values and also that of the negative predictive value and specificity value, formulas are give in Table 2. The harmonic mean is a measure of central tendency that takes into account the distance between the values and favours the lower value when calculating an average.

Figure 154 shows the harmonic mean (F_1 score) of the precision and recall values of each classification system at each iteration of the Adaboost algorithm. The bootstrap classifiers and bootstrap and feature-bagging classifiers averaged and smoothed the F_1 scores when compared to the individual classifier classification system. The bootstrap and feature-bagging classifier achieved a better F_1 score than the bootstrap classifiers. The bootstrap and feature-bagging classifiers successfully avoided the overfitting that occurred with the other classification systems using the validation dataset. The individual classifier achieved a slightly better F_1 score using outsample data compared to the other classification systems. However the F_1 score differences between each classification system are marginal.

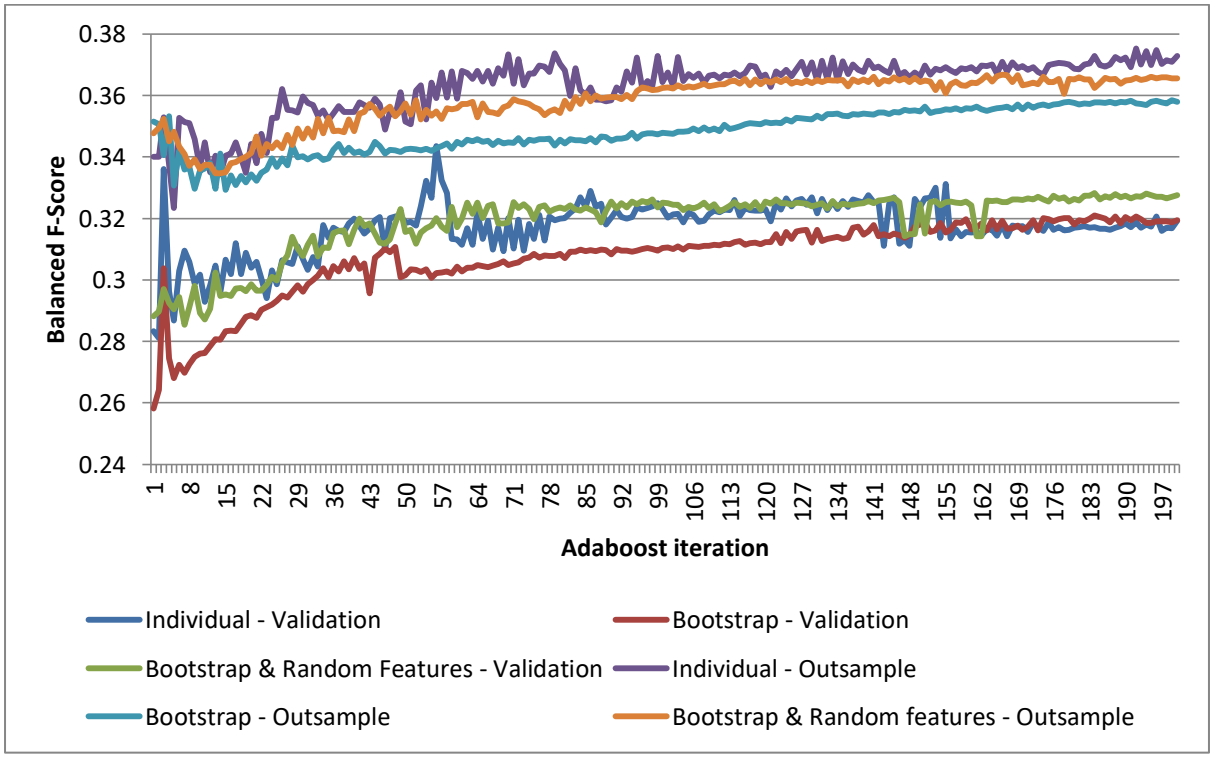


Figure 154: Balanced F-scores for each classification system classifying profitable technical analysis interpretations

Figure 155 shows the harmonic mean (F_1 score) of the negative predictive value and specificity values of each classification system at each iteration of the Adaboost algorithm. The bootstrap classifiers and the bootstrap and feature-bagging classifiers smoothed the F_1 scores when compared to the individual classifier classification system. The bootstrap and feature-bagging classification systems outperformed the other classification systems on both validation and outsample data. The overfitting that was present in the other classification systems using the validation data occurred once, but this classification system recovered from the overfitting.

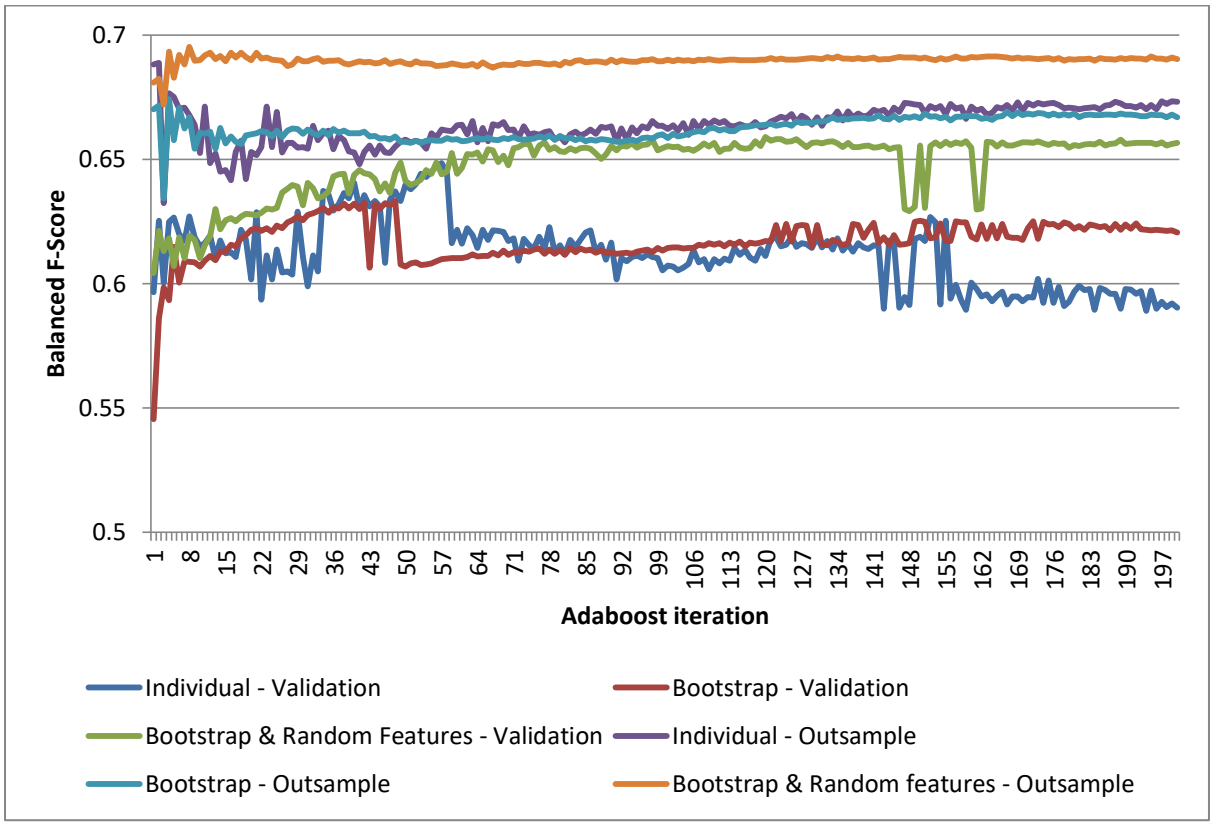


Figure 155: Balanced F-scores for each classification system classifying unprofitable technical analysis interpretations

8.4 Summary

Three classification systems were created to investigate whether the bootstrap aggregation technique and the feature-bagging technique can be used to help reduce the overfitting of classifiers.

When the classifiers were used on validation data, sharp decreases in performance (a sign of overfitting) were observed in the accuracy, precision, negative predictive value and specificity results. Interestingly no sharp decreases in performance were observed in the outsample dataset nor in the recall results when the classifiers were used on validation data. The bootstrap and feature-bagging classification system was the only classification system to recover from the overfitting observed in the graphs. The results of the classification system which used only the bootstrap aggregation technique saw only one occurrence of overfitting; this was the bootstrap classifier using USDCAD validation data. The results of the classification system which used neither of the techniques to reduce overfitting saw two occurrences of overfitting; these were the individual classifiers using AUDUSD and USDCAD validation data.

Unfortunately it is not clear which Adaboost iteration from the validation dataset results should be chosen for selecting a classifier with the best trade-off between bias and variance. The results show that choosing a classifier that has overfitted on validation data does not necessarily translate to inferior performance on outsample data. The accuracy, precision, negative predictive value and specificity values of the classifiers using outsample data converge irrespective of the overfitting that occurred in the validation dataset. Choosing an Adaboost iteration where the classifier's performance starts to converge is likely to yield a classifier with the best trade-off between bias and variance. Choosing a classifier after further iterations is likely to increase the chance of overfitting.

Classifying bad traders is one of the main focuses of this thesis, and correctly classifying and identifying unprofitable technical analysis interpretations is an important step towards building a system to detect bad traders. The classification system which used the bootstrap aggregation and feature-bagging technique performed best at classifying unprofitable technical analysis interpretations and identifying unprofitable technical analysis interpretations from the outsample dataset. At the 200th iteration of the Adaboost algorithm, the bootstrap and feature-bagging classification system achieved an F_1 score (derived from the negative predictive value and specificity) of 0.66 and 0.69 for validation data and outsample respectively. The individual Adaboost classifier classification system using validation and outsample data achieved 0.59 and 0.67. The bootstrap classification system using validation and outsample data achieved 0.62 and 0.67. The differences in F_1

score performance using outsample data are not evidently significant. If the validation dataset was used as a form of outsample data, instead of using the validation dataset to pick an 'optimal' iteration, then the bootstrap and feature-bagging classification system achieved a 0.07 and 0.04 F_1 score increase compared to the individual classification system and the bootstrap classification system respectively. The specificity results of each classification system saw the biggest differences. At the 200th iteration of the Adaboost algorithm, the specificity is 0.52 for the individual classifier classification system, 0.57 for the bootstrap classifier classification system and 0.64 for the bootstrap and feature-bagging classification system. The negative predictive value results show no significant difference between the classification systems on outsample data.

The precision values of the classification system using the bootstrap aggregation and feature-bagging technique were slightly better but not significantly better than the other classification systems. The recall values of individual classifiers without any techniques to reduce overfitting were slightly better but not significantly better than the other classification systems. Classification systems using the bootstrap aggregation and feature-bagging technique or just the bootstrap aggregation technique did not significantly improve the precision or recall values compared to the individual classifier classification system.

The classification systems are better at classifying and identifying unprofitable technical analysis interpretations than profitable technical analysis interpretations. This may be because some of the randomly created technical analysis interpretations are obviously bad. (An example might be one which is unprofitable on validation data, placed only two trades and has a large drawdown). Bad traders may deviate from their trading systems if they are experiencing tilt (fatigue, stress, greed, etc. which can introduce human errors) and the classification systems should be able to detect bad trading decisions via the trader's performance metrics.

It is important to observe that an approach that produces results better than random is encouraging, and so the results shown in this chapter indicate that the Adaboost classifier approach is suitable for classifying and identifying 'bad' technical analysis interpretations. The fact that it works well for classifying and identifying bad technical analysis interpretations matches well the goal of this thesis, and hence the Adaboost classifier with

bootstrap aggregation and feature-bagging is recommended (which achieved a 12% and 7% increase in specificity performance compared to individual Adaboost classifiers and the bootstrap classification systems). This constitutes a first model of traders, in which traders are represented simply as basing decisions on a single technical analysis interpretation. To extend this model, the next chapter will adopt a similar approach to exploring the classification of trading strategies composed of multiple technical analysis interpretations.

Chapter 9 – Classifying Trading Strategies using Adaboost and Dataset Balancing

The previous chapter investigated the performance of Adaboost-based classification systems in classifying profitable and unprofitable individual technical analysis interpretations. It was found that the best classification system used the *bootstrap aggregation* and *feature bagging* techniques, these comprised the only classification system that avoided the overfitting seen by the other tested classifications. It was also found that the validation dataset which is used to spot the overfitting of classifiers did not pinpoint an exact iteration at which to choose a classifier.

This chapter investigates the performance of classification systems for classifying profitable and unprofitable trading strategies that are more complex than individual technical analysis interpretations. The successful bootstrap aggregation and feature bagging techniques are again employed. An investigation is also presented into the performance differences between classification systems derived from a training dataset with roughly equal numbers of profitable and unprofitable trading strategies (termed 'balanced' datasets) and those derived from 'imbalanced' training datasets.

9.1 Datasets and Trading Strategy Generation

The previous chapter created classifiers using performance metrics derived from the trades of single technical analysis interpretations. In this chapter, performance metrics are derived from trading strategies which combine individual technical analysis interpretations in majority vote, as documented in Section 5.2.1. It is important to note that the size of a trading strategy is the number of technical analysis interpretations the trading strategy contains. The performance of classification systems using trading strategies that contain either 1, 3, 5, 11 or 21 technical analysis interpretations is investigated.

This chapter investigates the effect on the performance of classification system of differing balances between profitable and unprofitable strategies in the training datasets employed (The assessment of whether a trading strategy is profitable follows the approach to trading according to a strategy that is explained in Section 8.1.3) For imbalanced datasets (skewed towards either profitable or unprofitable strategies), technical analysis interpretations are

combined randomly in majority vote so that a variable proportion of profitable and unprofitable are created. For the balanced datasets, equal numbers of profitable and unprofitable trading strategies are created by combining technical analysis interpretations randomly in majority vote until equal numbers of profitable and unprofitable trading strategies are found.

The 7 foreign exchange markets and the segments used for training, validation and outsample market datasets are again those outlined in Section 8.1.2. A collection of 10,000 trading strategies is traded on the training, validation and outsample datasets to produce performance metrics that are used to train the classification system, choose an optimal classification system iteration and test the classification system.

To summarise, a collection of 10,000 trading strategies are created for each of the 7 foreign exchange markets and trading strategy sizes. The 10,000 trading strategies are either

1. Randomly created for the imbalanced training dataset configuration or
2. Are first traded on the training dataset to ensure that equal numbers of profitable and unprofitable trading strategies are created for the balanced training dataset configuration.

In total, 70 classification systems are created.

9.2 Imbalanced Training Dataset

Tables 25 and 26 show the maximum number of trading strategies that were classified as profitable across all iterations of the Adaboost algorithm for each classification system on imbalanced training, validation and outsample datasets (each dataset contains the same 10,000 trading strategies). In other words, the number of trading strategies classified as profitable is considered at each of the 200 iterations of the Adaboost algorithm, and the maximum of these numbers is taken, regardless of whether those classifications are consistent or confirmed at the end of the run. The cells that are highlighted in orange indicate that fewer than 200 trading strategies were classified as profitable during the creation of the classification system, identifying collections that may lead to unreliable results.

The classification systems using NZDUSD and USDCHF training data derived from trading strategies of size 11 and 21 did not classify many or any trading strategies as profitable, except for the system using NZDUSD outsample data derived from trading strategies of size 11 which classified 1,131 trading strategies as profitable. Classification systems for other markets using outsample data derived from trading strategies of those sizes found a much larger proportion to be profitable. The classification systems using NZDUSD and USDCHF training data derived from trading strategies of size 5 classified only 153 and 75 trading strategies as profitable, respectively. These two classification systems classified significantly fewer trading strategies as profitable using validation and outsample data compared to the classification systems using the other markets and validation and outsample datasets derived from trading strategies of size 5; these all classified above 7,900 trading strategies as profitable.

As it is possible for classification systems to classify all trading strategies as profitable during the initial iterations of the Adaboost algorithm, values in the table that are close to 10,000 do not indicate that the classification system has classified almost all trading strategies as profitable at the end of 200 iterations. At a later iteration of the Adaboost algorithm, the classifier may learn to correct the misclassifications of the previous classifier at the previous iteration. Very low values for the number of strategies classified as profitable stand out as unusual, and may indicate that the classifier is failing to learn how to classify trading strategies as profitable (for a given market, strategy size, and the corresponding collection of 10,000 trading strategies).

	1			3			5		
	Training	Validation	Outsample	Training	Validation	Outsample	Training	Validation	Outsample
AUDUSD	2068	2757	2281	4612	6942	4647	7982	9168	9044
EURUSD	5007	5326	3941	9333	9849	8295	9613	9997	9970
GBPUSD	6990	7787	3342	9783	9975	9906	9942	9996	9997
NZDUSD	957	1429	904	453	2253	2280	153	769	1392
USDCAD	3202	4214	2559	8071	9111	9978	9271	9992	9997
USDCHF	884	2312	1495	233	8416	1796	75	932	338
USDJPY	8600	8920	8902	9908	9871	9974	9983	9999	9996

Table 25: Maximum number of trading strategies classified as profitable for each trading strategy size, market and for imbalanced datasets (1 of 2)

	11			21		
	Training	Validation	Outsample	Training	Validation	Outsample
AUDUSD	9136	9544	9651	9286	7210	9346
EURUSD	9834	10000	10000	9786	9978	10000
GBPUSD	9994	10000	10000	9999	10000	9998
NZDUSD	11	168	1131	0	1	107
USDCAD	9741	9696	9998	9733	9008	9990
USDCHF	22	47	9	0	66	2
USDJPY	9998	9977	9997	9997	9929	10000

Table 26: Maximum number of trading strategies classified as profitable for each trading strategy size, market and for imbalanced datasets (2 of 2)

Tables 27 and 28 show the maximum numbers of trading strategies that were classified as unprofitable across all iterations of the Adaboost algorithm. The cells that are highlighted in red indicate that fewer than 200 trading strategies were classified as unprofitable during the creation of the classification system. The cells that are highlighted orange indicate that fewer than 200 trading strategies were classified as profitable as copied from Tables 25 and 26 (to assist in identifying classification systems that did not classify enough profitable or unprofitable trading strategies).

The classification systems using EURUSD, GBPUSD and USDJPY training data derived from trading strategies of size 11 or 21 did not classify many trading strategies as unprofitable in either the training dataset or outsample dataset. For systems using GBPUSD validation data derived from trading strategies of size 11 and 21, only a few trading strategies were classified as unprofitable. The classification systems using GBPUSD and USDJPY training data derived from trading strategies of size 5 also classified fewer than 200 trading strategies respectively as unprofitable, and only a maximum of 170 and 94 trading strategies as

unprofitable using the validation dataset. Similarly at size 5, the classification system using USDJPY outsample data only classified a maximum of 123 trading strategies as unprofitable.

	1			3			5		
	Training	Validation	Outsample	Training	Validation	Outsample	Training	Validation	Outsample
AUDUSD	6018	7929	8283	4692	4746	6904	2521	2940	4363
EURUSD	4282	5560	7894	1693	2221	5205	566	847	1533
GBPUSD	2993	3077	7083	557	1125	4196	71	170	1478
NZDUSD	9149	9288	9310	9802	9640	8660	9983	9972	9723
USDCAD	6226	7950	7944	2492	2555	3776	1192	759	744
USDCHF	9473	9258	9263	9956	4411	8663	9999	9856	9997
USDJPY	2333	3211	3272	289	361	227	120	94	123

Table 27: Maximum number of trading strategies classified as unprofitable for each trading strategy size, market and for imbalanced datasets (1 of 2)

	11			21		
	Training	Validation	Outsample	Training	Validation	Outsample
AUDUSD	1164	2329	4479	498	4114	1278
EURUSD	5	427	0	130	849	0
GBPUSD	32	47	61	4	33	185
NZDUSD	9997	9985	9380	10000	9998	9996
USDCAD	394	456	87	339	2288	20
USDCHF	9999	10000	9998	10000	10000	10000
USDJPY	1	375	1	23	1274	0

Table 28: Maximum number of trading strategies classified as unprofitable for each trading strategy size, market and for imbalanced datasets (2 of 2)

The numbers of trading strategies classified as profitable or unprofitable is tiny in a few classification systems, in particular for systems that use a training dataset derived from trading strategies containing 11 or 21 technical analysis interpretations. The classification systems in the tables denoted by the highlighted cells may have been created from a training dataset with a significant imbalance in the number of profitable and unprofitable trading strategies, which in turn may have affected the ability of the systems to learn their classifications.

To investigate this further, Table 29 shows the numbers of trading strategies that were profitable, unprofitable and neither profitable nor unprofitable for each dataset (using the method of trading described in Section 8.1.3). The training, validation and outsample datasets derived from trading strategies of size 1, 3 or 5 contain healthy numbers of both profitable and unprofitable trading strategies, but less consistency for sizes 11 and 21.

		1			3			5			11			21		
		Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither
AUDUSD	Training	3170	4590	2240	4858	3877	1265	5794	3323	883	6782	2509	709	7269	2101	630
	Validation	2497	7390	113	2579	7413	8	2434	7563	3	2398	7598	4	2492	7508	0
	Outsample	3006	6916	78	2646	7353	1	2573	7427	0	2036	7963	1	1463	8536	1
EURUSD	Training	4979	3521	1500	7409	1973	618	8302	1312	386	9140	694	166	8863	923	214
	Validation	2312	7624	64	1320	8677	3	878	9122	0	395	9605	0	184	9816	0
	Outsample	2104	7859	37	1148	8850	2	804	9195	1	245	9755	0	51	9949	0
GBPUSD	Training	6087	3079	834	7786	2032	182	8409	1533	58	9158	836	6	9544	455	1
	Validation	2184	7663	153	1469	8522	9	1050	8946	4	403	9597	0	138	9862	0
	Outsample	3798	5938	264	4190	5785	25	4473	5524	3	3424	6576	0	1822	8176	2
NZDUSD	Training	2458	7122	420	1998	7955	47	1588	8406	6	854	9145	1	536	9464	0
	Validation	2936	6815	249	3747	6243	10	4276	5721	3	5274	4726	0	5707	4291	2
	Outsample	5171	4625	204	5721	4269	10	5898	4101	1	7012	2988	0	7799	2198	3
USDCAD	Training	3988	4206	1806	5948	3249	803	6906	2681	413	7952	1802	246	8331	1496	173
	Validation	4110	5792	98	4886	5114	0	5171	4828	1	5706	4294	0	5578	4421	1
	Outsample	3325	6419	256	4531	5455	14	5209	4788	3	5323	4676	1	4993	5007	0
USDCHF	Training	1513	8368	119	884	9115	1	546	9454	0	182	9817	1	31	9969	0
	Validation	2074	7761	165	1366	8631	3	907	9093	0	384	9616	0	121	9879	0
	Outsample	2301	7542	157	1959	8035	6	1646	8354	0	1169	8829	2	758	9242	0
USDJPY	Training	6383	3281	336	7888	2055	57	8492	1491	17	8982	1016	2	8991	1006	3
	Validation	2141	7810	49	1506	8490	4	1156	8844	0	574	9425	1	285	9714	1
	Outsample	4109	5840	51	4817	5177	6	5427	4570	3	6244	3753	3	6983	3017	0

Table 29: Number of profitable, unprofitable and neither profitable nor unprofitable trading strategies for each scenario

Now focusing on those classification systems that classified fewer than 200 trading strategies as profitable or unprofitable (from Tables 25, 26, 27 and 28), Table 30 shows again the numbers of demonstrably profitable, unprofitable and neither profitable nor unprofitable trading strategies (as copied from Table 29) with coloured cells linking back to the classification results. The highlighted cells indicate whether the classification system classified fewer than 200 trading strategies as profitable (orange) or fewer than 200 trading strategies as unprofitable (red). The orange highlighted cells for the USDCHF dataset show that the number of unprofitable trading strategies outweighs the number of profitable trading strategies. The training datasets which are highlighted orange have more unprofitable trading strategies than profitable. Similarly, the training datasets which are highlighted red have more profitable than unprofitable.

The imbalance in the proportion of profitable and unprofitable trading strategies in the training dataset seems to have produced classification systems that favour one classification over the other. For example, the classification system using GBPUSD training data derived from trading strategies of size 21 contained 9,544 profitable trading strategies and 455 unprofitable trading strategies, and fewer than 200 trading strategies were classified as unprofitable during training for all iterations of the Adaboost algorithm. The corresponding validation and outsample datasets contained 138 and 1,822 profitable trading strategies and 9,862 and 8,176 unprofitable trading strategies respectively. The classifier classified fewer than 200 trading strategies as unprofitable even though the number of unprofitable trading strategies significantly outweighed the number of profitable trading strategies. This may support the view that a lack of balance of profitable and unprofitable trading strategies impacts on the ability of a classifier to learn effectively, leading to experimentation with balanced datasets in the next section.

		5			11			21		
		Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither
EURUSD	Training				9140	694	166	8863	923	214
	Validation				395	9605	0	184	9816	0
	Outsample				245	9755	0	51	9949	0
GBPUSD	Training	8409	1533	58	9158	836	6	9544	455	1
	Validation	1050	8946	4	403	9597	0	138	9862	0
	Outsample	4473	5524	3	3424	6576	0	1822	8176	2
NZDUSD	Training	1588	8406	6	854	9145	1	536	9464	0
	Validation	4276	5721	3	5274	4726	0	5707	4291	2
	Outsample	5898	4101	1	7012	2988	0	7799	2198	3
USDCAD	Training				7952	1802	246	8331	1496	173
	Validation				5706	4294	0	5578	4421	1
	Outsample				5323	4676	1	4993	5007	0
USDCHF	Training	546	9454	0	182	9817	1	31	9969	0
	Validation	907	9093	0	384	9616	0	121	9879	0
	Outsample	1646	8354	0	1169	8829	2	758	9242	0
USDJPY	Training	8492	1491	17	8982	1016	2	8991	1006	3
	Validation	1156	8844	0	574	9425	1	285	9714	1
	Outsample	5427	4570	3	6244	3753	3	6983	3017	0

Table 30: Number of profitable, unprofitable and neither profitable nor unprofitable trading strategies for each scenario

9.3 Balanced Training Dataset

Tables 31 and 32 show the maximum numbers of trading strategies that were classified as profitable across all iterations of the Adaboost algorithm (as in Section 9.2, but for balanced datasets). The imbalanced training dataset produced some classification systems that classified fewer than 200 trading strategies as profitable (Tables 25 and 26). The balanced training dataset produced classifiers that classified at least 620 trading strategies as profitable across all iterations of the Adaboost algorithm on training, validation and outsample data.

	1			3			5		
	Training	Validation	Outsample	Training	Validation	Outsample	Training	Validation	Outsample
AUDUSD	6226	5199	5040	5393	5093	3563	5483	5191	3289
EURUSD	7984	9780	9856	5972	7315	7758	6185	7633	8073
GBPUSD	5248	5498	4973	4788	3641	2184	4764	3802	1333
NZDUSD	4763	8726	8038	5425	8973	8679	5212	9019	8453
USDCAD	7487	5914	5340	7969	8088	8182	7223	7832	8347
USDCHF	5015	8818	7795	5612	8259	9285	5714	8428	9561
USDJPY	5764	5215	5644	5928	6920	5012	6518	7615	5959

Table 31: Maximum number of trading strategies classified as profitable for each trading strategy size, market and for balanced datasets (1 of 2)

	11			21		
	Training	Validation	Outsample	Training	Validation	Outsample
AUDUSD	5842	3630	6125	5116	2200	4783
EURUSD	6154	9828	9994	4671	9456	10000
GBPUSD	5120	3723	2349	5259	4733	4838
NZDUSD	5506	8984	7905	5475	8403	8226
USDCAD	5770	4515	4944	4729	620	854
USDCHF	5605	6627	8752	5352	5979	7439
USDJPY	5068	6160	9895	4299	1359	9998

Table 32: Maximum number of trading strategies classified as profitable for each trading strategy size, market and for balanced datasets (2 of 2)

Tables 33 and 34 show the maximum numbers of trading strategies that were classified as unprofitable across all iterations of the Adaboost algorithm. While the imbalanced training datasets produced some classification systems that classified fewer than 200 trading strategies as unprofitable (Tables 25 and 26), the balanced training datasets produced classifiers that classified at least 1097 trading strategies as unprofitable across all iterations of the Adaboost algorithm on training, validation and outsample data. The only exception was the classification system using EURUSD outsample data, derived from trading strategies of size 21, which saw the maximum number of trading strategies classified as unprofitable as 1.

	1			3			5		
	Training	Validation	Outsample	Training	Validation	Outsample	Training	Validation	Outsample
AUDUSD	4865	5457	5912	5492	5163	6949	5377	5418	7406
EURUSD	8564	8886	8214	5272	4757	6222	5105	4379	5590
GBPUSD	5353	7680	9339	6108	7153	9507	5636	7305	9405
NZDUSD	6123	5014	2744	5726	3967	2891	5828	5096	3647
USDCAD	4937	6189	6872	5465	6483	6754	4735	4871	4501
USDCHF	5272	5366	5285	4910	3767	4078	5140	2796	3582
USDJPY	4802	6646	6361	4317	6422	6752	4385	5491	6340

Table 33: Maximum number of trading strategies classified as unprofitable for each trading strategy size, market and for balanced datasets (1 of 2)

	11			21		
	Training	Validation	Outsample	Training	Validation	Outsample
AUDUSD	4855	8066	7376	5191	9058	7247
EURUSD	5365	3508	1097	5874	2297	1
GBPUSD	5305	6855	8369	5252	6682	8147
NZDUSD	5299	3701	3544	6193	7717	9686
USDCAD	5516	8670	8537	5975	9836	9962
USDCHF	5011	4728	3295	4758	4936	3533
USDJPY	5660	6394	3499	6684	9900	1794

Table 34: Maximum number of trading strategies classified as unprofitable for each trading strategy size, market and for balanced datasets (2 of 2)

Figure 156 shows the maximum numbers of profitable and unprofitable trading strategies classified across all iterations of the Adaboost algorithm on validation data and also on outsample data, for every classification system, plotting the numbers for balanced datasets in blue and imbalanced in red. The results show that the classification systems trained using the imbalanced training dataset favoured labelling most of the trading strategies as either profitable or unprofitable. The balanced training dataset however tended not to label all, or nearly all, trading strategies as profitable or unprofitable at any iteration.

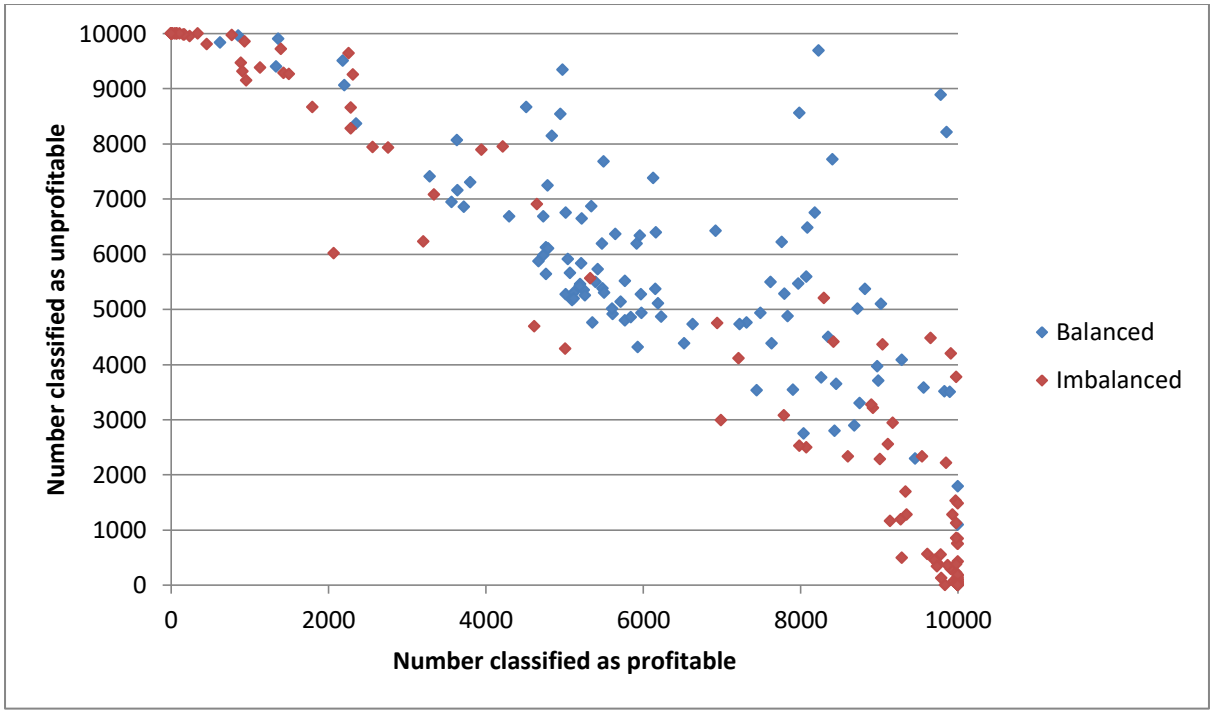


Figure 156: The maximum number of trading strategies classified as profitable and unprofitable for every classification using the validation and outsample datasets

Table 35 shows the numbers of trading strategies that were profitable, unprofitable and neither profitable nor unprofitable for each balanced dataset (a counterpart to Table 29 for imbalanced datasets, again using the method of trading explained in Section 8.1.3). The numbers of profitable trading strategies are lower in the EURUSD, GBPUSD, USDCHF and USDJPY validation datasets derived from trading strategies of size 21, and for the EURUSD outsample dataset derived from trading strategies of size 21. The EURUSD outsample dataset derived from trading strategies of size 21 contained only 11 profitable trading strategies, the lowest found. Note that the classification system using the EURUSD outsample dataset derived from trading strategies of size 21 classified only one trading strategy as unprofitable (Table 34) when 9,989 of the trading strategies were unprofitable.

The trading behaviour of a trading strategy with 1 technical analysis interpretation is likely to be consistent in the future compared to the trading behaviour of 21 technical analysis interpretations in majority vote. Trading strategies that consist of 21 technical analysis interpretations contributing to the final decision of the trading strategy may be too complex and behave differently in the future because of changing market conditions. Additionally, good combinations of technical analysis algorithms may be

present in the trading strategy but can also contain randomly performing technical analysis interpretations that add noise to the majority vote decision mechanism of the trading strategy.

Comparing the classification systems built from balanced and imbalanced datasets, the classification systems built using the imbalanced dataset tended to produce classification systems that classified a very low percentage of the validation and outsample dataset as a particular classification (profitable or unprofitable) and the a very high percentage as the other classification. This is especially true for classification systems that used a training dataset of performance metrics derived from trading strategies with large numbers of technical analysis interpretations in majority vote.

		1			3			5			11			21		
		Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither	Profitable	Unprofitable	Neither
AUDUSD	Training	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0
	Validation	2995	6887	118	2866	7129	5	2506	7491	3	2318	7679	3	2254	7746	0
	Outsample	3605	6347	48	3614	6385	1	3563	6435	2	3688	6309	3	3714	6285	1
EURUSD	Training	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0
	Validation	2514	7424	62	1837	8162	1	1464	8536	0	667	9333	0	241	9758	1
	Outsample	1784	8213	3	1047	8953	0	678	9322	0	148	9852	0	11	9989	0
GBPUSD	Training	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0
	Validation	2215	7773	12	1543	8455	2	1056	8943	1	405	9595	0	79	9921	0
	Outsample	3814	6184	2	4554	5445	1	4868	5132	0	4181	5819	0	2373	7627	0
NZDUSD	Training	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0
	Validation	2788	7198	14	3629	6370	1	4245	5753	2	4330	5670	0	2938	7062	0
	Outsample	5454	4534	12	5375	4622	3	5729	4268	3	5680	4319	1	5852	4147	1
USDCAD	Training	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0
	Validation	5091	4880	29	6097	3901	2	6820	3180	0	8062	1937	1	8685	1315	0
	Outsample	3699	6249	52	4618	5382	0	5340	4660	0	5791	4208	1	6021	3979	0
USDCHF	Training	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0
	Validation	2751	7198	51	2285	7711	4	1920	8077	3	1147	8853	0	621	9379	0
	Outsample	3113	6766	121	3075	6921	4	2820	7179	1	2490	7509	1	1919	8080	1
USDJPY	Training	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0	5000	5000	0
	Validation	1904	8091	5	1424	8576	0	1045	8955	0	308	9692	0	136	9864	0
	Outsample	4437	5489	74	5440	4556	4	6296	3703	1	7510	2489	1	8187	1812	1

Table 35: Number of profitable, unprofitable and neither profitable nor unprofitable trading strategies for each scenario

9.4 Classification System Results

This section explores the differences between the performance of classification systems that are created from an imbalanced training dataset and a balanced training dataset. The explored classification systems are outlined in Section 9.1. The figures in this section average the binary classification performance results of the 7 foreign exchange markets.

Some of the classification systems created from an imbalanced training dataset (Section 9.2) classified only a few or none of the trading strategies as profitable or unprofitable. This has led to misleading results where classification systems have skewed the average performance values by producing continuous zeros or ones or producing values that oscillate because only a few trading strategies were classified. In such cases, the classification system produced 'divide by zero' values which cannot be averaged. The performance results from the imbalanced training datasets derived from trading strategies containing 11 or 21 technical analysis interpretations are removed from all graphs except for the accuracy graphs.

Figure 157 shows the average accuracy of classification systems derived from imbalanced training datasets. The classification systems achieved better average accuracy values using outsample data compared to using validation data. The classification systems derived from trading strategies of size 1 outperformed the others on both validation and outsample datasets. For classification systems using validation data derived from trading strategies with three or more technical analysis interpretations, average accuracy drops to 0.4 or below, whereas for the outsample datasets all are above 0.4. All classification systems using outsample data, except those using outsample data derived from trading strategies of size 1, converge to roughly the same average accuracy value of between 0.38 and 0.4.

The average accuracy results show no instances of overfitting that lead to a decrease in accuracy. Slight dips in accuracy performance are observed in the classification systems using validation data derived from trading strategies of size 1 and 3, followed by recovery. For size 1, the analysis in Chapter 8 established the use of bootstrap aggregation and feature bagging to be responsible for this recovery, and it seems reasonable to speculate that these methods may be the cause of the recovery in size 3 as well.

The classification systems using outsample data derived from trading strategies of sizes 1 and 3 performed better than random with accuracy values of 0.6 and 0.54 at the 200th iteration of the Adaboost algorithm.

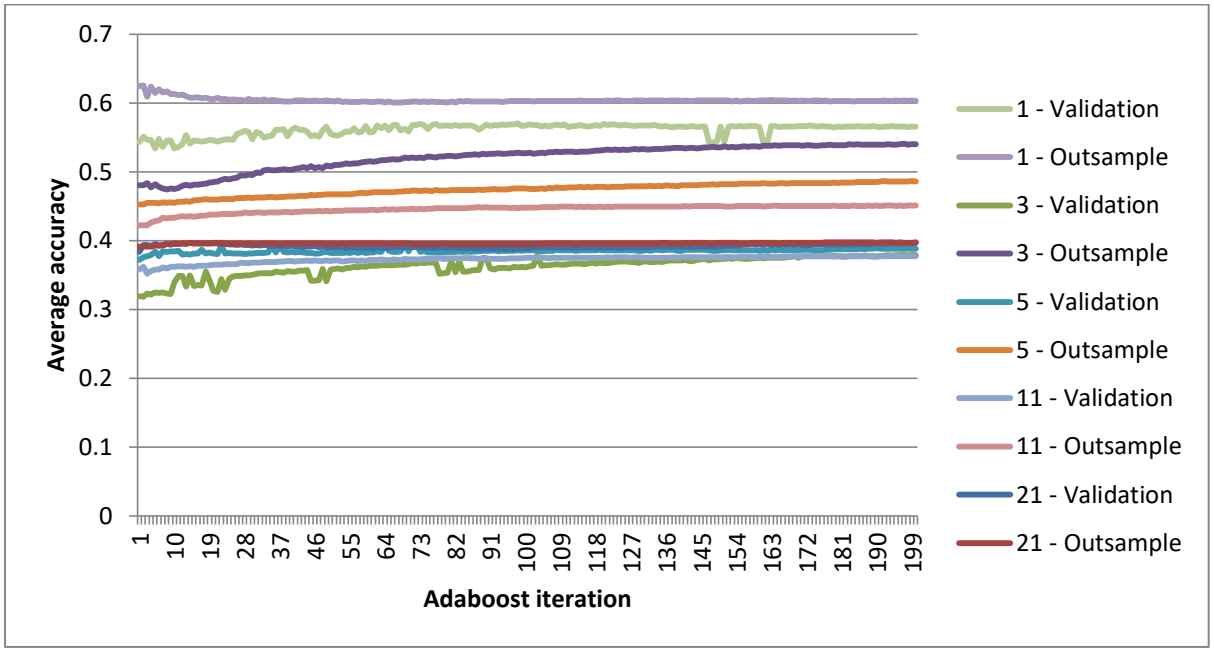


Figure 157: The average accuracy of classification systems using an imbalanced training dataset

Figure 158 shows the average accuracy of classification systems derived from balanced training datasets. The average accuracy values of classification systems derived from trading strategies of all sizes are closer together than the classification systems using an imbalanced training dataset, with all results being nearly random or worse than random by the 200th iteration; the lowest results occur for sizes 11 and 12 on validation data, and the system for size 21 on outsample data was close behind.

While the use of balanced datasets seems to have avoided classifications that perform extremely badly, the best accuracy results occur for trading strategies containing only one technical analysis interpretation which achieved a higher average accuracy value using a slightly imbalanced training dataset (Table 29). The numbers of profitable and unprofitable trading strategies in the imbalanced and balanced training datasets are not significantly different for this size. In contrast, classification systems using an imbalanced training dataset derived from trading strategies of size 11 or 21 had larger inequalities, compared to the

balanced datasets, in the numbers of profitable and unprofitable trading strategies, accounting for the greater differences in average accuracy value.

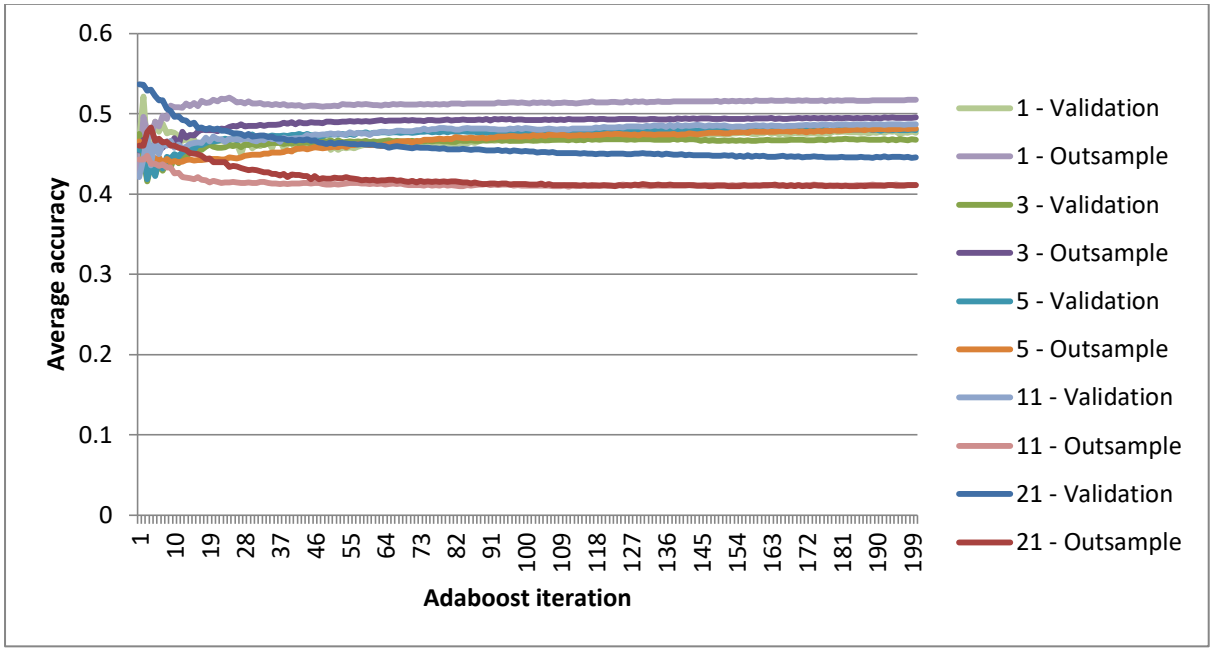


Figure 158: The average accuracy of classification systems using a balanced training dataset

Figure 159 shows the average precision of classification systems derived from imbalanced training datasets. All average precision values are worse than random. The average precision of classification systems derived from trading strategies containing one technical analysis interpretation outperformed the other classification systems.

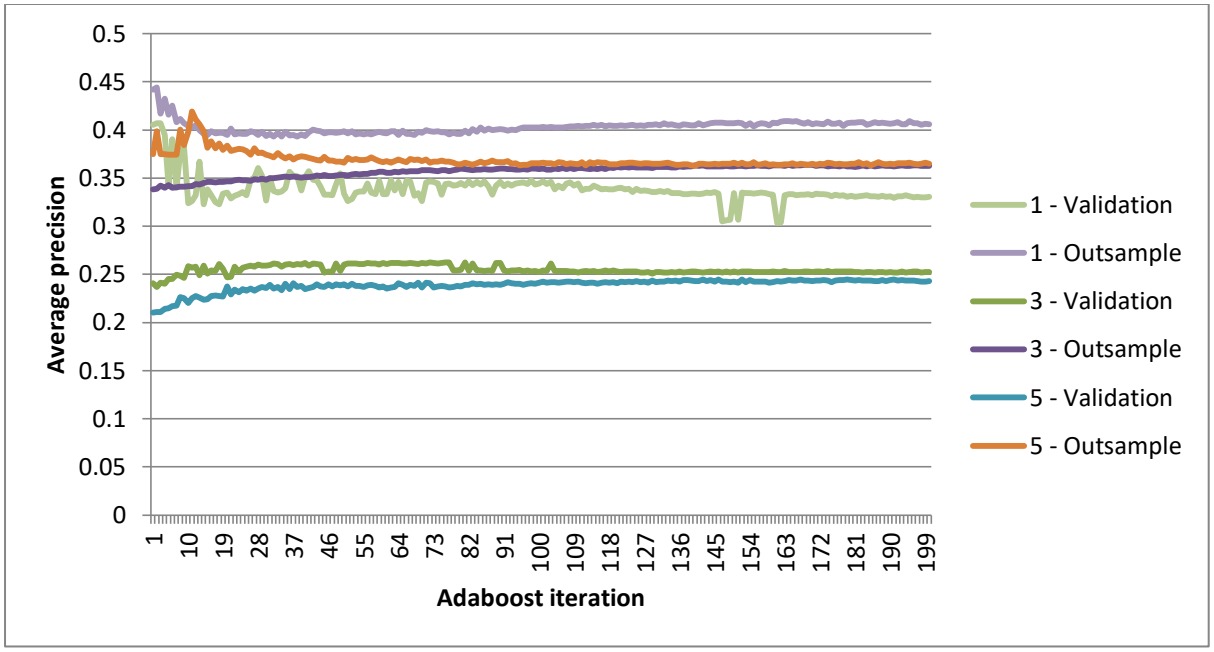


Figure 159: The average precision of classification systems using an imbalanced training dataset

Figure 160 shows the average precision of classification systems derived from balanced training datasets, and again all average precision values are also worse than random and again the classification systems derived from trading strategies containing one technical analysis interpretation outperformed the other classification systems. The classification systems using a balanced training dataset achieve an average precision that clusters around 0.25 and 0.35 average precision for validation and outsample data respectively.

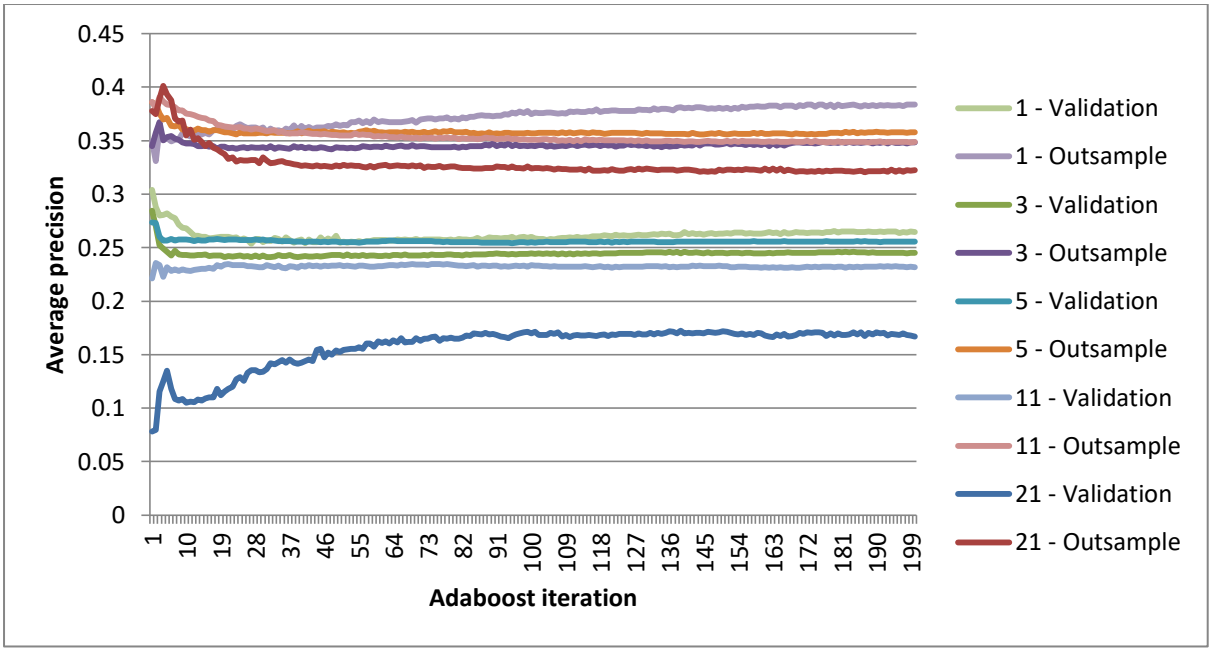


Figure 160: The average precision of classification systems using a balanced dataset

The average recall measure results are shown in Figure 161 for classification systems derived from an imbalanced training dataset. The systems using datasets derived from trading strategies with one technical analysis interpretation have a lower average recall value than the classification systems using datasets derived from trading strategies of sizes 3 and 5. Classification systems using trading strategies of size 3 and 5 achieve better than random performance and are better at identifying profitable trading strategies when using validation data than when using outsample data.

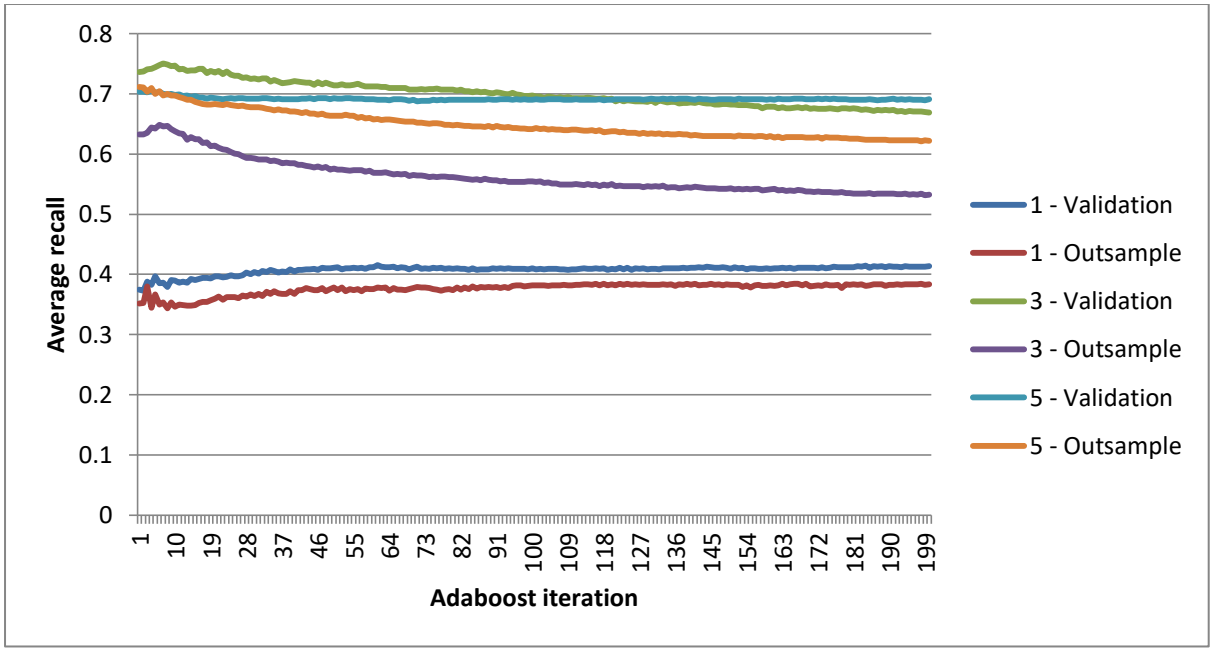


Figure 161: The average recall of classification systems using an imbalanced dataset

The corresponding recall results for the balanced training datasets are shown in Figure 162. The average recall values of the classification system using datasets derived from trading strategies that contain one technical analysis interpretation are approximately 0.5. The same classification systems using an imbalanced training dataset perform far worse. However, the imbalanced training datasets which contained trading strategies of size 3 or 5 outperformed all classification systems using a balanced dataset. This may be because the classification systems trained using the imbalanced dataset are more likely to be bias towards a particular classification such as profitable.

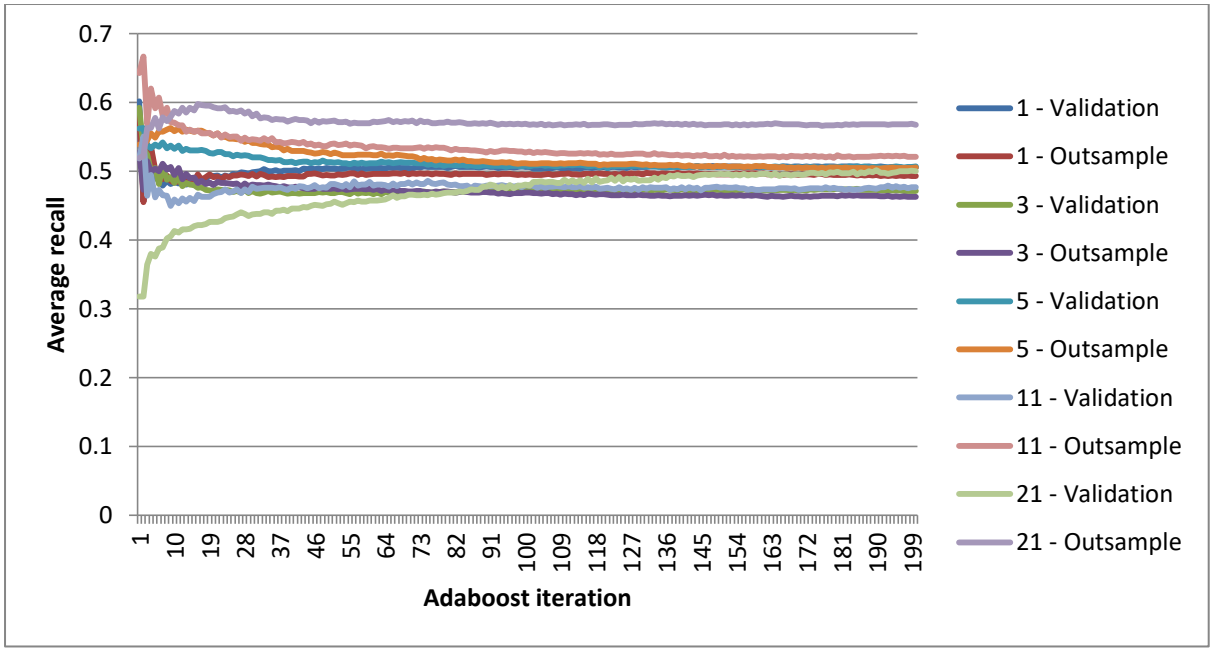


Figure 162: The average recall of classification systems using a balanced dataset

Figure 163 shows the average negative predictive value of classification systems derived from imbalanced training datasets, and all systems can be seen to perform better than random. The average negative predictive value is higher in the systems using validation data rather than outsample data.

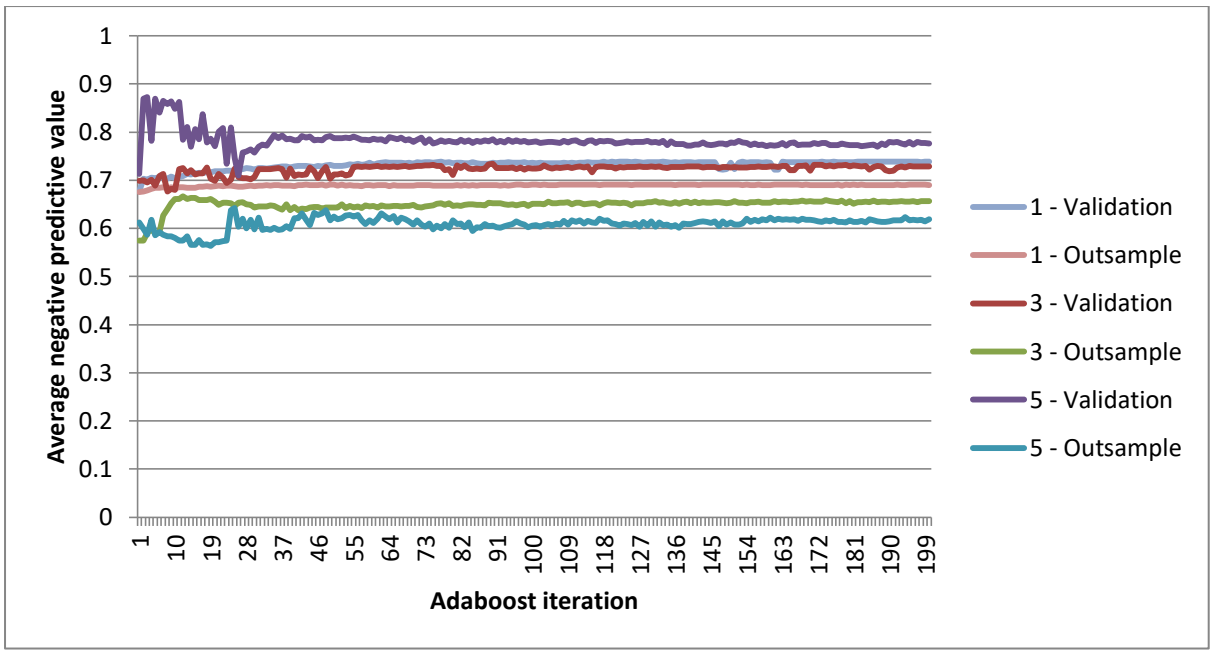


Figure 163: The average negative predictive value of classification systems using an imbalanced dataset

The corresponding average negative predictive values for systems derived from balanced training datasets are presented in Figure 164. These values are higher for classification systems derived from validation data than for those derived from outsample data; the values for those two categories appear to cluster close together.

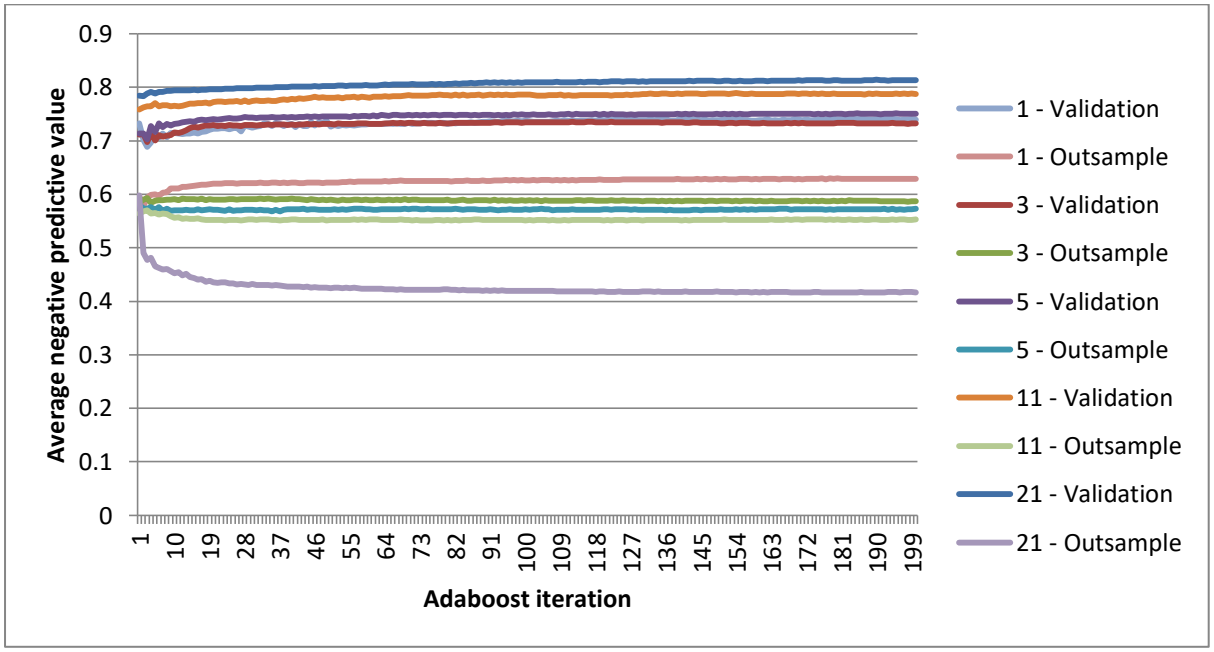


Figure 164: The average negative predictive value of classification systems using a balanced dataset

Figure 165 shows the average specificity values of classification systems derived from imbalanced training datasets. The average specificity values from classification systems using datasets derived from trading strategies with 1 technical analysis interpretation identified 64% and 73% of the unprofitable trading strategies in the validation and outsample datasets, respectively, at the 200th iteration. At that iteration, the average specificity values of classification systems using datasets derived from trading strategies of size 5 and from the validation dataset derived from trading strategies of size 3 identified less than 38% of the unprofitable technical analysis interpretations in their respective datasets. Also at the 200th iteration, the classification systems using outsample data derived from trading strategies of size 3 achieved an average specificity value of 0.52, indicating only random identification of unprofitable trading strategies.

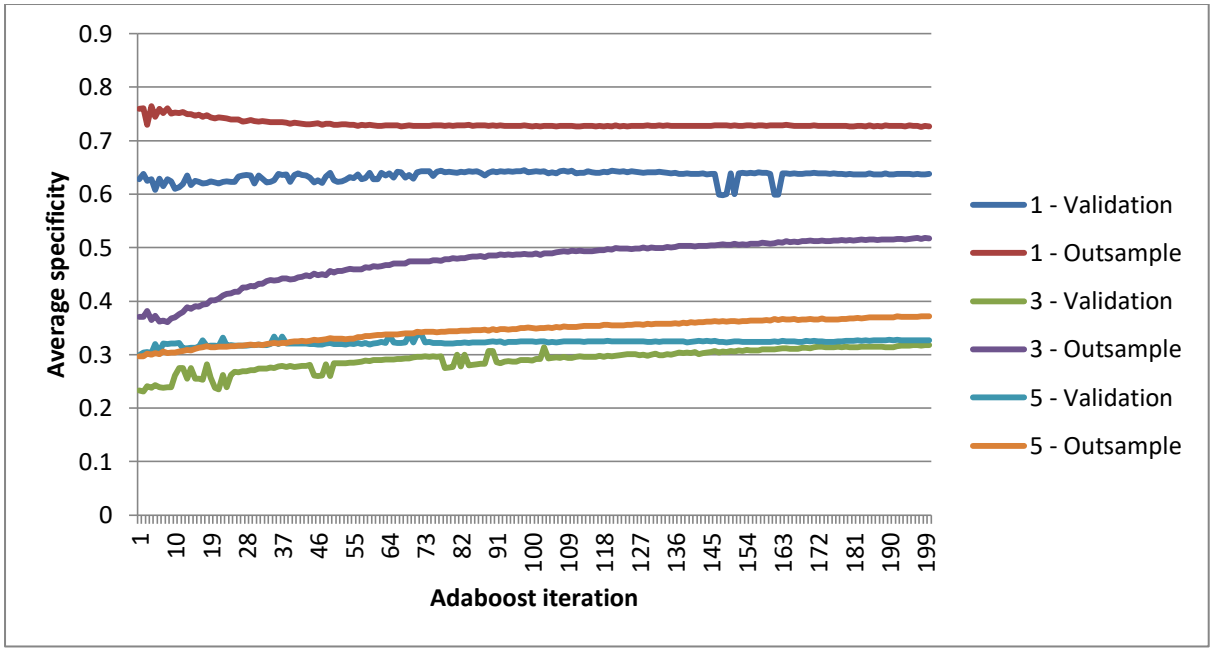


Figure 165: The average specificity of classification systems using an imbalanced dataset

Figure 166 shows the average specificity of classification systems derived from a balanced training dataset. At the 200th iteration of the Adaboost algorithm all average specificity values are 0.52 or less; the values here are closer together than for the imbalanced dataset.

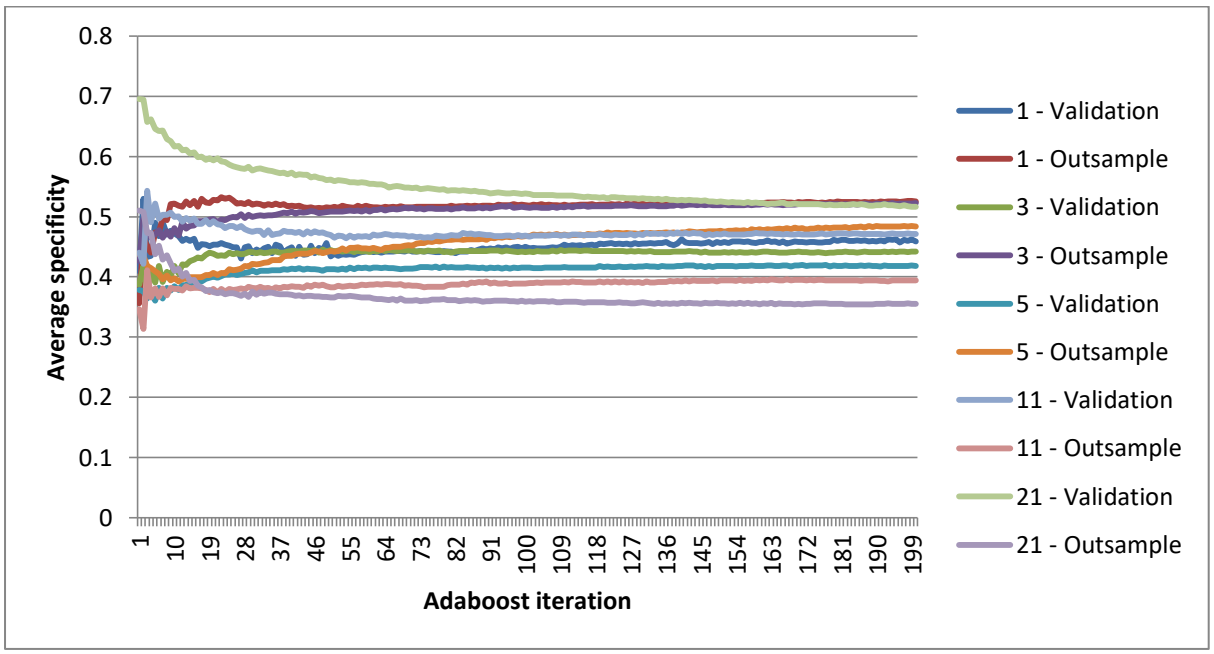


Figure 166: The average specificity of classification systems using a balanced dataset

9.5 Summary

The main focus of this thesis is to detect poor traders by using the performance metrics obtained from trading strategies. The average negative predictive value and average specificity results are important in summarising a classification system's performance on unprofitable trading strategies. For the balanced datasets, classification systems achieve average negative predictive values of roughly 0.6 and 0.75 for the validation dataset and outsample dataset at the 200th iteration of the Adaboost algorithm, and these classification systems are particularly good at classifying trading strategies as unprofitable. Unfortunately the average specificity results show that these classification systems identified only 53% of the unprofitable trading strategies at best. The validation dataset is used to identify the optimal iteration before a classification system overfits. However, the validation dataset has little bearing on the classification system's subsequent performance on outsample data.

The binary classification performance results of classification systems derived from a balanced dataset produce more reliable performance results compared to classification systems derived from an imbalanced dataset. Additionally, the performance results of classification systems using balanced training datasets from trading strategies of different sizes were more similar compared to classification systems trained from a imbalanced dataset. Only one classification system derived from a balanced dataset incorrectly classifies almost all trading strategies as profitable. Many of the classification systems derived from an imbalanced dataset, in particular datasets derived from trading strategies of sizes 11 and 21, classify almost all the training dataset as either entirely profitable or entirely unprofitable. These classifiers went on to classify the trading strategies in the validation and outsample datasets as the same classification. These flawed classification systems were removed from the reported performance results to avoid reporting misleading results. The classification systems using the balanced datasets tend to have similar average performance values across validation and outsample datasets.

Classification systems derived from an imbalanced dataset are unable to produce reliable classifiers from training datasets derived from trading strategies containing 5, 11 or 21 technical analysis interpretations. Those using training datasets derived from trading strategies containing 1 or 3 technical analysis interpretations do not seem to have the same flaw of classifying a future dataset as all profitable or all unprofitable. This suggests that

trading strategies containing many technical analysis interpretations may be too complex to produce trading strategies that behave similarly in the future. These trading strategies may contain good combinations of technical analysis interpretations. However there may be technical analysis interpretations that add noise to the majority vote decision of the trading strategy.

While the balanced datasets produce the most reliable classification systems and perform well at classifying unprofitable trading strategies, the results fall short of a sufficient mechanism for detecting poor traders on outsample data. The average negative predictive value for trading strategies of size 1, 3 and 5 are 0.63, 0.59 and 0.57 respectively at the 200^{th} iteration at the Adaboost algorithm on outsample data which is better than random performance. The validation dataset is used to find an optimal iteration to choose a classification system that begins to converge. As financial markets are non-stationary (Schmitt, Chetalova, Schäfer, & Guhr, 2013) and change over time, the validation dataset which continues on from the market data used to create the training dataset is more likely to exhibit market conditions and behaviour of the training dataset. If the validation dataset can be used as a form of outsample dataset then the classification system achieves average negative predictive values between 0.73 and 0.8.

The training datasets used so far are all derived from trading strategies using the same historical period of market data. Market conditions may be highly volatile or the market may be in a long term downtrend or uptrend, and hence the validation and outsample periods of market data may show different market conditions. To reduce this problem, a classification system should be created using a training dataset containing many market datasets or different periods of market data. A training dataset derived from different markets will be explored in the next chapter.

Chapter 10 – Classification Systems using Multimarket Training Datasets

The previous chapter investigated the performance of Adaboost-based classification systems at classifying profitable and unprofitable trading strategies. Each classification system used the same bootstrap aggregation and feature bagging techniques to avoid overfitting. The experiments varied the number of technical analysis interpretations used in each trading strategy, and also investigated the performance of the classification systems using both imbalanced and balanced datasets. It was shown that imbalanced datasets had some tendency, particularly for strategies of larger size, to produce classification systems that incorrectly classified all trading strategies as profitable or all as unprofitable; balanced datasets reduced that tendency. In particular, classification systems created from an imbalanced training dataset derived from trading strategies containing 11 or 21 technical analysis interpretations were likely to have a greater difference between the number of profitable and unprofitable trading strategies. These systems also produced classifications that mimicked those made on the training dataset (for example, the classification system using a imbalanced training dataset derived from trading strategies of size 21 trading the GBPUSD market, had an overwhelming number of profitable trading strategies in the training dataset, then classified the validation and outsample dataset as also being predominately profitable, despite future datasets actually containing the opposite proportion of profitable and unprofitable trading strategies). It was also shown that the use of balanced datasets reduced the above problem, but in both balanced and imbalanced dataset cases it is important to note that the proportions of profitable and unprofitable strategies observed in the training dataset may not reflect the proportions in the validation and outsample dataset.

As concluded from the previous chapter, market conditions may be highly volatile or the market may be in a long term downtrend or uptrend. Market conditions may therefore differ between the training dataset and future validation and outsample datasets. To attempt to reduce this problem, this chapter investigates classification systems that use a balanced dataset consisting of multiple sources of market data over the same time period. This chapter also investigates the use of trading strategies composed of different numbers

of technical analysis interpretations, and compares their results with classification systems derived from a balanced dataset using one market (from the previous chapter).

10.1 Multimarket Datasets

Previous experiments (Chapters 8 and 9) used a training dataset consisting of performance metrics derived from technical analysis interpretations or trading strategies trained on a single market. In this chapter the training dataset is built from performance metrics derived from trading strategies using all seven of the foreign exchange market datasets, AUDUSD, EURUSD, GBPUSD, NZDUSD, USDCAD, USDCHF or USDJPY. Each of the datasets contains 10,000 randomly created trading strategies, using equal numbers of trading strategies derived from each of the foreign exchange market datasets. The strategies derived from each market are also balanced - selecting equal numbers of profitable and unprofitable strategies (establishing profitability on the training data relating to the segment of data for that market, by trading in the way described in Section 8.1.3). Each trading strategy is given a particular market from which to derive performance metrics, and uses the same market to calculate future performance on the validation and outsample datasets.

10.2 Classification System Results

Figure 167 shows the average accuracy performance of classification systems trained on multimarket data when used on the validation dataset, for trading strategies of different sizes. At the 200th iteration of the Adaboost algorithm, the classification system using the performance metrics derived from trading strategies of size 1 achieved an accuracy value of 0.6. For sizes greater than 1, however, lower accuracy values are achieved, ranging from 0.45 to 0.52. The equivalent results for the outsample dataset are presented in Figure 168. At the 200th iteration of the Adaboost algorithm, the classification systems using performance metrics derived from trading strategies of size 1, 3 and 5 achieved accuracy values of 0.56, 0.56 and 0.54 respectively. Those for size 7, 9, 11 and 21 achieved lower accuracy values (less than 0.47). These accuracy results suggest that performance metrics derived from smaller size trading strategies create better classification systems, and specifically of size 1 in relation to the validation dataset.

Comparing the above accuracy results with those for classification systems which use only one market to derive performance metrics, in the previous chapter (Figure 158), it is clear

that the outsample accuracy values are better in the multimarket classification system. The validation accuracy values are similar for all classification systems using performance metrics derived from trading strategies of size 3 or above. At the 200th iteration of the Adaboost algorithm, the classification system using performance metrics derived from trading strategies of size 1 from many markets achieved an accuracy value of 0.6, whereas the same classification systems built from single markets achieved only an average accuracy value of 0.48.

The validation dataset showed no optimal iteration for choosing the best classification system which would in turn lead to an optimal classification on outsample data.

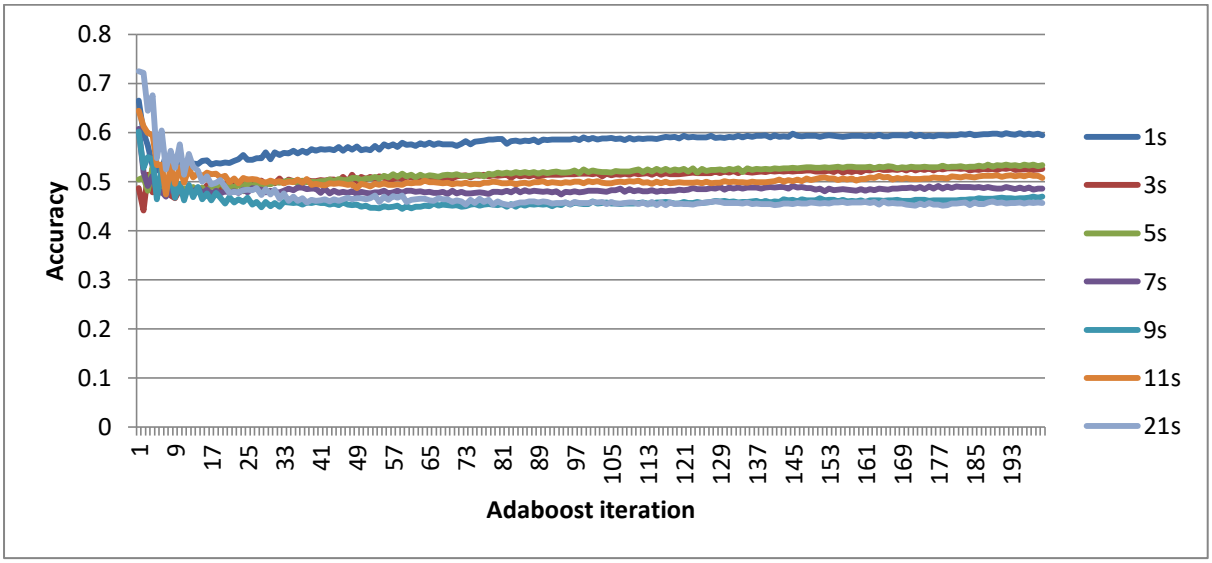


Figure 167: Accuracy of classification systems of different trading strategy sizes trained on multimarket data and used on validation data

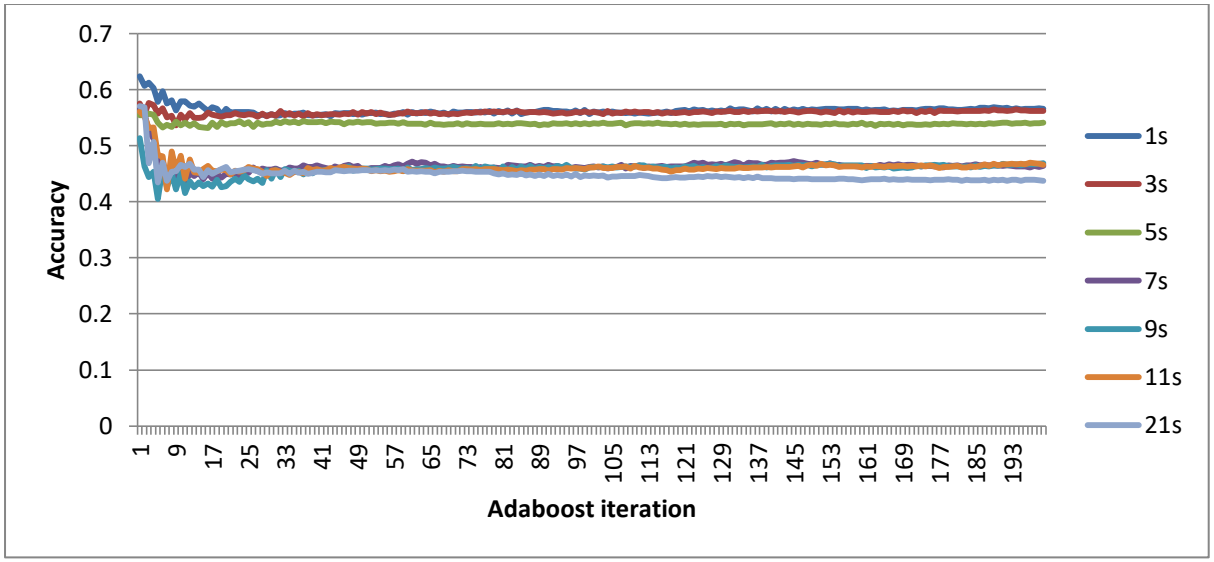


Figure 168: Accuracy of classification systems of different trading strategy sizes trained on multimarket data and used on outsample data

Figure 169 shows the average precision performance of classification systems trained on multimarket data when used on the validation dataset, for trading strategies of different sizes. At the 200th iteration of the Adaboost algorithm, the systems of size 1 achieved an average precision value of 0.35 which is higher than the results for systems using trading strategies of sizes greater than 1. The corresponding results on the outsample dataset are shown in Figure 170. At the 200th iteration of the Adaboost algorithm, the classification systems using performance metrics derived from trading strategies of size 1, 3 and 5 achieved precision values of 0.43, 0.45 and 0.45 respectively. Those using trading strategies of size 7, 9, 11 and 21 achieved lower precision values (between 0.37 and 0.4). This again supports the view that performance metrics derived from trading strategies using fewer technical analysis interpretations create better classification systems. Using the precision results obtained from the validation dataset, classification systems appear to be more successful when derived from trading strategies of size 1.

These outsample precision values are higher than the average precision values of the classification systems using individual markets (Figure 160 in the previous chapter). At the 200th iteration of the Adaboost algorithm, all precision values of the classification systems using performance metrics derived from multiple markets are above 0.37; when using performance metrics derived from single markets, all average results are below 0.37 except for those from trading strategies of size 1 (average precision value 0.38). However, the latter

result is still bettered by the average precision value of 0.43 achieved in the classification systems using the same trading strategy configuration but performance metrics derived from multiple markets. Again, use of the multiple market dataset achieves better results but, in terms of precision, all the classification systems using validation and outsample data perform worse than random. The validation dataset showed no optimal iteration for choosing an optimal classification system on outsample data.

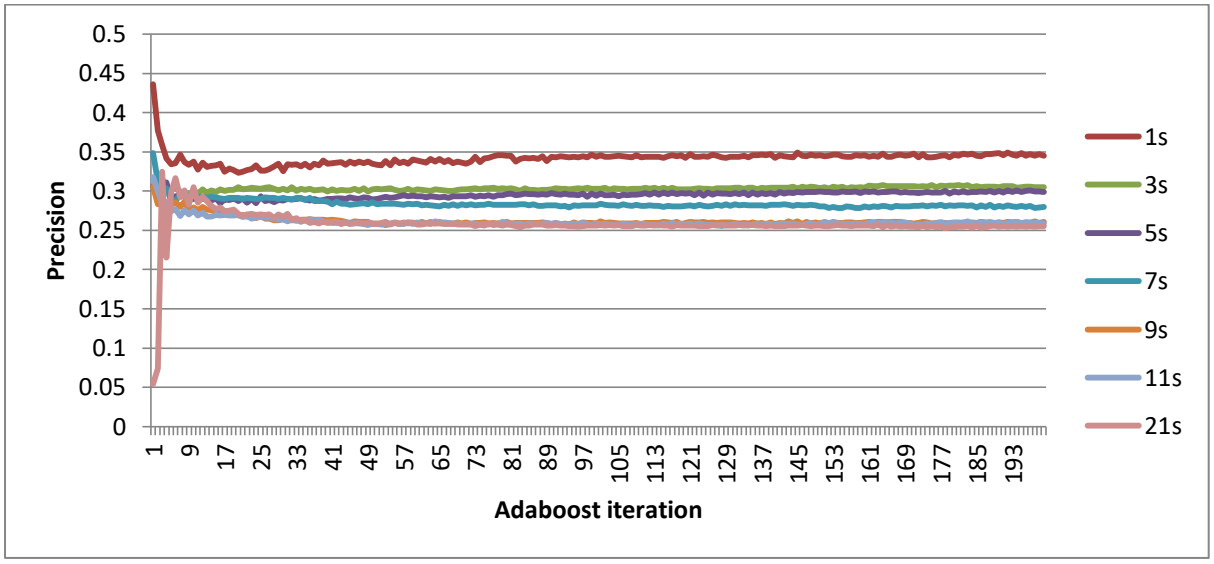


Figure 169: Precision of classification systems of different trading strategy sizes trained on multimarket data and used on validation data

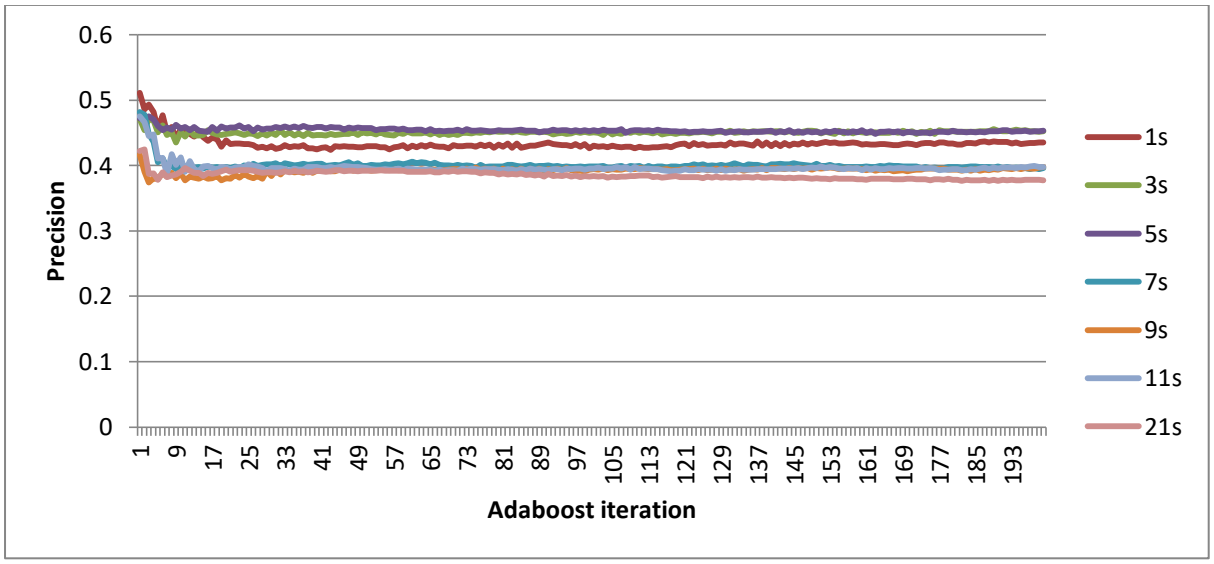


Figure 170: Precision of classification systems of different trading strategy sizes trained on multimarket data and used on outsample data

Figures 171 and 172 show the average recall performance of classification systems trained on multimarket data when used on the validation dataset and the outsample dataset respectively, for trading strategies of different sizes. During the initial iterations of the Adaboost algorithm, the precision values peak between the 10^{th} and 20^{th} iteration before each declining and converging to individual long-term values. This suggests that the validation dataset can be used to find an optimal classification on outsample data.

The systems using performance metrics derived from trading strategies of size 21 achieved a higher recall value on both the validation and outsample datasets than the other classification systems. The recall performance of classification systems using the outsample dataset increases with the number of technical analysis interpretations per trading strategy. For validation data, average results are lowest when the performance metrics are derived from trading strategies of size 1, and highest when the performance metrics are derived from trading strategies of size 21.

Comparing these recall results with the average recall results of classification systems each of which used only one market to derive performance metrics (see Figure 162 in the previous chapter), the recall values are similar. However, the recall value of the classification system using multiple markets has an optimum between the 10^{th} and 20^{th} iteration for selecting a classification system before the classification system overfits.

At the 10^{th} iteration of the Adaboost algorithm, the recall values are 0.55 or above and 0.5 or above for all classification systems on validation data and outsample data respectively.

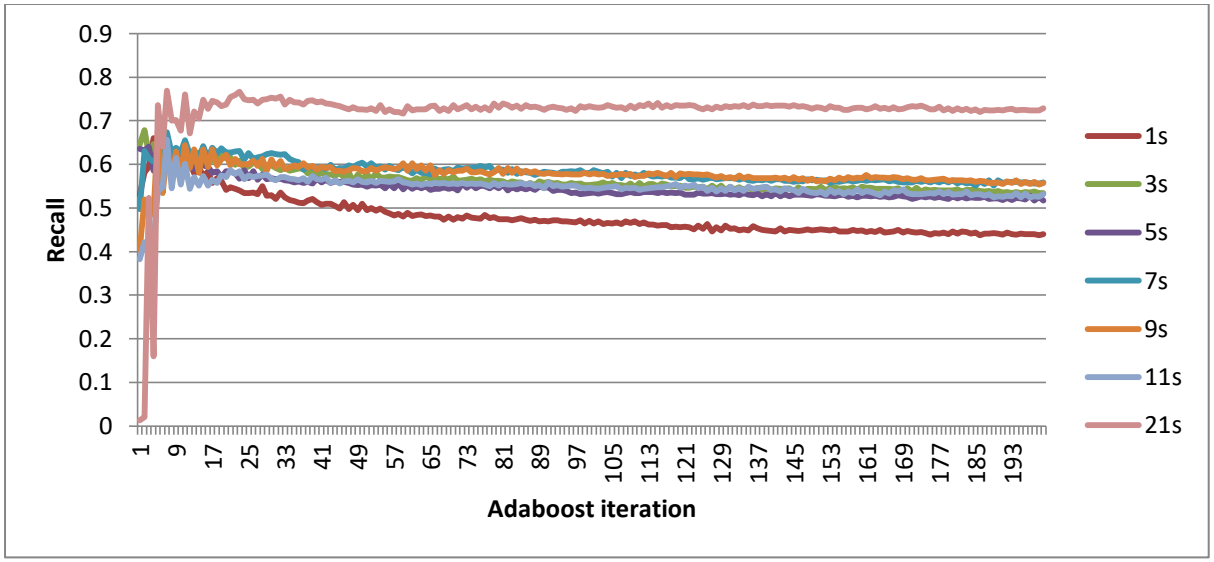


Figure 171: Recall of classification systems of different trading strategy sizes trained on multimarket data and used on validation data

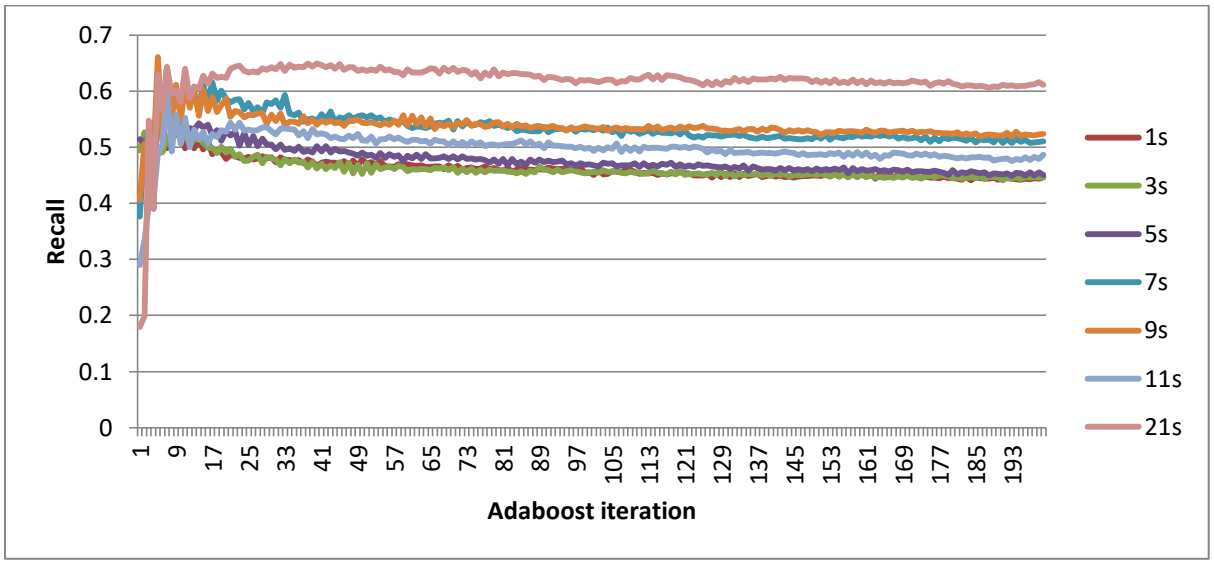


Figure 172: Recall of classification systems of different trading strategy sizes trained on multimarket data and used on outsample data

Figure 173 shows the average negative predictive value performance of classification systems trained on multimarket data when used on the validation dataset for trading strategies of different sizes. The values for each classification system are at their highest at around the 10th iteration of the Adaboost algorithm, after which they each either decline or converge. Systems using performance metrics derived from trading strategies of size 21 outperformed the other classification systems in this measure, with a value of roughly 0.85 around the 10th iteration of the Adaboost algorithm. All of the negative predictive values across all 200 iterations of the Adaboost algorithm are above 0.72. Figure 174 presents the

equivalent results when applied to the outsample dataset. The average negative predictive values converge within a few iterations, unlike those of classification systems using validation data. The optimal classification system iteration chosen from the negative predictive value results obtained from validation data did not increase negative predictive value performance, in that all systems achieved about the same negative predictive value across all iterations of the Adaboost algorithm.

Comparing these results with those of classification systems each of which use only one market to derive performance metrics (Figure 164 in the previous chapter), the negative predictive values at the 200th iteration of the Adaboost algorithm are similar when used on outsample data. The lowest average negative predictive value were the systems using performance metrics derived from trading strategies of size 21 that we using individual market datasets. The equivalent classification systems using multiple markets behaved similarly to the classification systems for other sizes. The negative predictive values of classification systems using validation data behaved similarly. Systems using performance metrics derived from trading strategies of size 11 and 21, however, did better when the performance metrics were sourced from multiple markets instead of a single market.

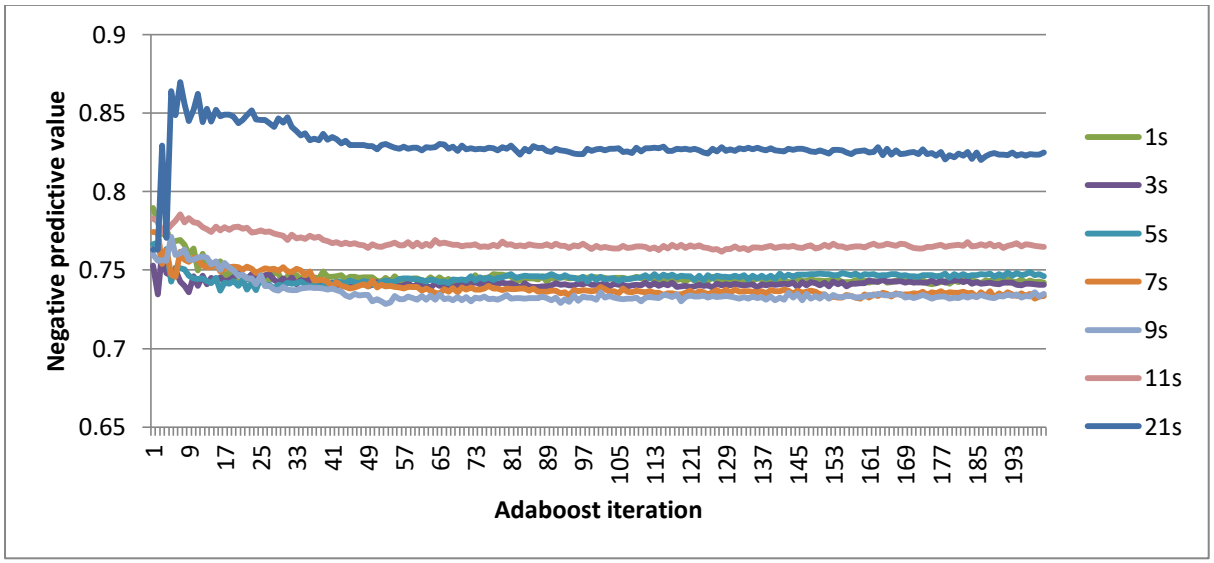


Figure 173: Negative predictive value of classification systems of different trading strategy sizes trained on multimarket data and used on validation data

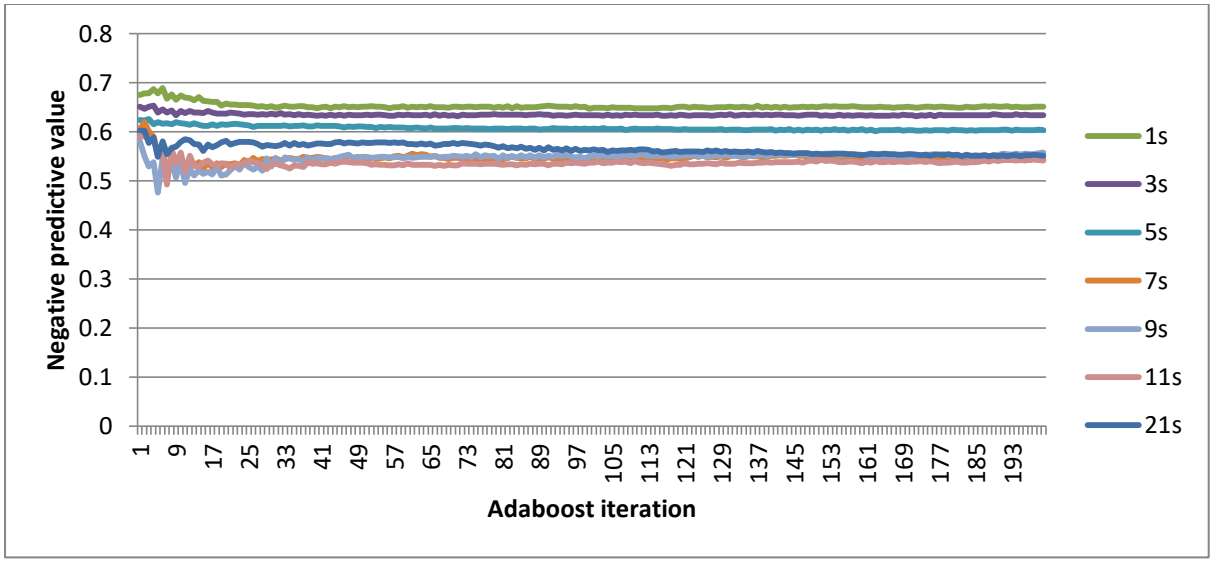


Figure 174: Negative predictive value of classification systems of different trading strategy sizes trained on multimarket data and used on outsample data

Lastly, Figure 175 shows the average specificity performance of classification systems trained on multimarket data when used on the validation dataset, for trading strategies of different sizes. At the 200th iteration of the Adaboost algorithm, the specificity value of the classification system using performance metrics derived from trading strategies of size 1 is 0.66 whereas the other classification systems have specificity values less than 0.54. Figure 176 shows the corresponding results for the outsample dataset. The specificity values of the classification systems using performance metrics derived from trading strategies of size 1, 3 and 5 achieve specificity values of 0.64, 0.64 and 0.6 respectively; the systems for sizes 7, 9, 11 and 21, however, performed worse than random by this measure, obtaining specificity values less than 0.45. The classification systems using performance metrics derived from trading strategies of size 21 had the lowest specificity value (0.32) at the 200th iteration of the Adaboost algorithm.

Comparing these specificity values with the average specificity values of classification systems each of which use only one market to derive performance metrics (Figure 166 in the previous chapter), the specificity values of classification systems using trading strategies of size 1, 3 and 5 on outsample data were higher when using performance metrics built from multiple markets. At the 200th iteration of the Adaboost algorithm, classification systems using performance metrics obtained from multiple markets achieved specificity

values of 0.6 or slightly higher, whereas the average specificity values are 0.52 or less for classification systems using performance metrics obtained from a single market.

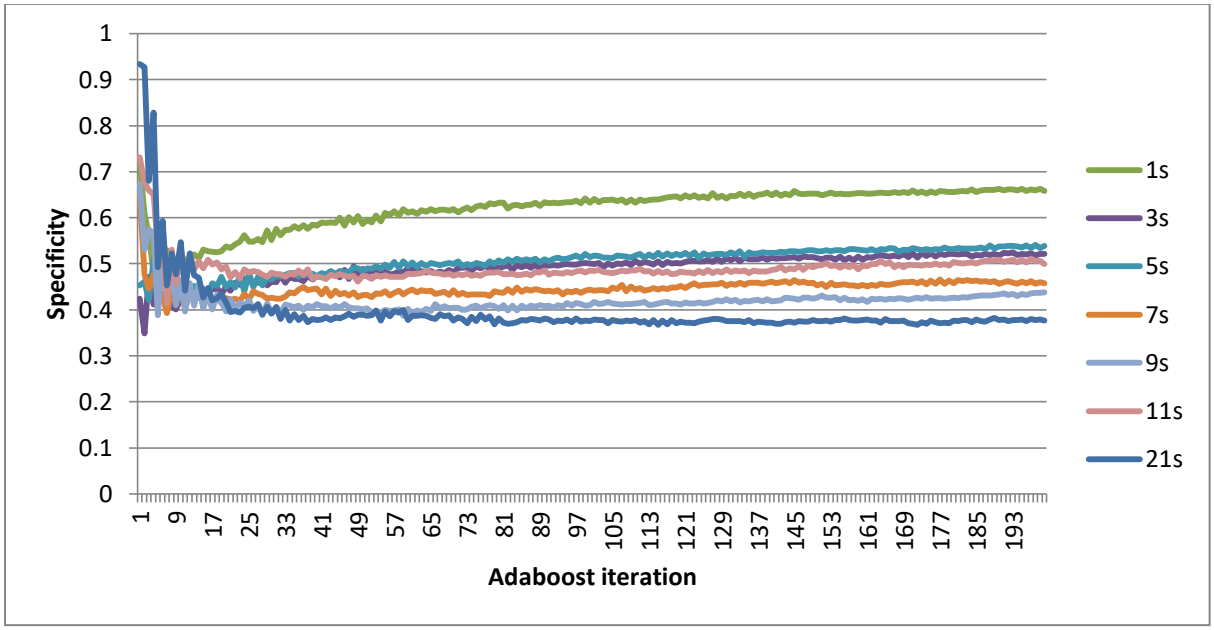


Figure 175: Specificity of classification systems of different trading strategy sizes trained on multimarket data and used on validation data

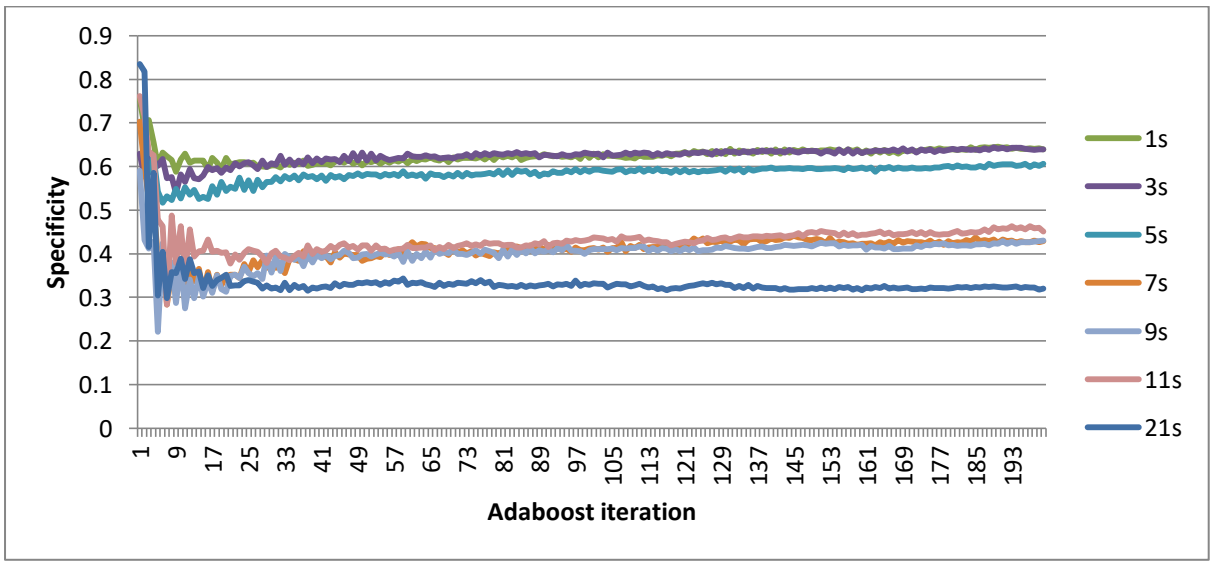


Figure 176: Specificity of classification systems of different trading strategy sizes trained on multimarket data and used on outsample data

10.3 Summary

The purpose of this chapter was to investigate whether classification systems derived from performance metrics obtained from multiple markets would outperform performance metrics derived from a single market. Trading strategies use a segment of market data to produce performance metrics for each dataset, and so it is possible that a market segment consists of only one market condition, for example high volatility or long-term downtrend. A classification system trained on one market may be biased towards classifying trading strategies as profitable if the trading strategy mimics the market conditions in that single market segment. Thus, the classification system would be unable to classify trading strategies if market conditions change.

Classification systems using performance metrics that were sourced from multiple markets are shown to outperform classification systems using performance metrics sourced from only a single market. The classification systems using multiple markets produced the same or higher accuracy, precision, recall, negative predictive value and specificity values; although the negative predictive value results were marginally improved, this binary classification metric shows the best results overall, for both multimarket and single market data. These results suggest, as hoped, that using multiple markets over the same time period increases the chances that a variety of market conditions are observed by the classification system, so producing a classifier that has greater generality.

The difference in the performance of classification systems using performance metrics derived from trading strategies of different sizes is interesting. In this chapter, performance metrics derived from trading strategies of size 1, which is equivalent to a technical analysis interpretation model, were better at classifying future performance. Compared to the other classification systems, the systems using performance metrics derived from trading strategies of size 1 had the best accuracy, precision and specificity values in the validation dataset, and the best negative predictive value in the outsample dataset. This classification system was also amongst the best in accuracy, precision and specificity in the outsample dataset. A negative predictive value of 0.75 and 0.65 was obtained on validation and outsample data respectively, but the classification system was amongst the worst for recall performance in both the validation and outsample datasets.

The purpose of this thesis is to derive a method by which a 'bad' trader may be classified as such. Throughout the thesis, it is assumed that trading strategies can be used as a model of traders, and that a ‘bad’ trader is an unprofitable trader. The trading strategy model is somewhat simplistic and does not include money management components such as position sizing, take profit levels and stop loss levels. Traders may also employ very sophisticated trading systems. Ideally, real trader performance metrics are needed to train the classification systems. The notion that a ‘bad’ trading strategy is unprofitable may be false, in that e.g. current market conditions may be exceptional, or an otherwise good strategy may be inappropriately applied or insufficiently funded. Therefore, an alternative criterion such as significant negative returns or a Sharpe ratio below 1 might be a better decision metric and boundary. The next chapter will explore various classification system decision metrics and boundaries. Nevertheless, this and the previous chapter have established the benefits of using balanced datasets comprising data from multiple markets.

Chapter 11 – Exploring Trading Strategy Categorisations

The previous chapter compared the performance of classification systems derived from performance metrics obtained from trading strategies using seven different foreign exchange markets, with those relating to a single foreign exchange market. It was found that the use of multimarket data improved performance. Results were produced for trading strategies containing different numbers of technical analysis interpretations. It was found that classification systems using trading strategies containing only one technical analysis interpretation performed better than classification systems using trading strategies of three interpretations or more.

Previous chapters categorized a trading strategy as ‘bad’ if the trading strategy was not profitable and as ‘good’ if the trading strategy was profitable. The performance metric values of a trading strategy are subject to variance as the performance is evaluated on an arbitrary segment of market data. Additionally, a trading strategy may buy and sell randomly which should, on average, create a return of zero (as the experiments do not include transaction costs) within some standard deviation. Adaboost may overfit due to the existence of this variance and noise. This chapter will explore the use of other categorizations for ‘good’ and ‘bad’ trading strategies by changing the classification metric and threshold value used in classification systems that define the decision boundary.

11.1 Trading Strategy Categorisations

The Adaboost algorithm is a supervised learning algorithm which uses a training dataset. This training dataset is made up of performance metrics of trading strategies obtained from the first segment of market data (Section 8.1.2) and labelled using the classification system’s categorisation metric and value on the second segment of market data. The classification systems produced in experiments in the previous chapters attempt to categorise trading strategies as 'profitable' and 'unprofitable'. This is achieved by using the return metric (Section 8.1.3) as the label with a threshold such that if the return is above zero the trading strategy is profitable and if below zero the trading strategy is unprofitable. That is, the Adaboost algorithm would learn the training dataset and would create a decision boundary that attempts to distinguish between profitable and unprofitable trading strategies.

The categorisation of trading strategies can be performed in several ways. Categorisation based upon whether a trading strategy is profitable or unprofitable may be subject to noise due to the performance of randomly generated trading strategies, and because the performance metrics are dependent on the character of the market segment. The following experiments change the labels within the datasets, in order to investigate the performance of classification systems under different categorisation metrics and values. The following experiments investigate the binary categorisations outlined in Table 36.

Label metric	Label value
Expected Payoff	-100, -500, 0, 100 and 500
Profit Factor	0.25, 0.5, 0.75, 1, 1.5 and 2
Return	-10000, -1000, 0, 1000 and 10000
Sharpe Ratio	-0.5, 0, 0.5, 1
Sortino Ratio	-0.5, 0, 0.5, 1
Trade Success Rate	0.3, 0.4, 0.5, 0.6 and 0.7

Table 36: The classification system categorisation performance metric and values explored

The return, profit factor and expected payoff performance metrics (defined in Section 4.8) all represent the profitability of trading strategies but in different ways. The Sharpe ratio and Sortino ratio metrics (defined in Sections 4.1 and 4.2 respectively) both represent how much return is expected per unit risk; the Sortino ratio is a variant which uses downside risk instead of the Sharpe ratio's standard deviation of returns (a proxy for risk). Both the Sharpe ratio and Sortino ratio metrics are used when maximising an objective function in an evolutionary process (Chapter 6). The trade success rate metric (defined in Section 4.8) represents how often the trading strategy's buy and sell predictions are profitable.

The performance of each classification system is reported using the accuracy, precision, recall, negative predictive value and specificity values. Any trading strategy above any of the investigated categorisation metrics and values in Table 36 can naturally be labelled 'good' therefore all precision and recall results show the classification system's performance at classifying and identifying trading strategies above the classification system's categorisation metric value.

11.2 Training, Validation and Outsample Dataset Categorisations

The balanced training dataset consists of performance metrics obtained from 10,000 trading strategies each of which uses one of the seven foreign exchange market datasets (as

described in Section 10.1). Equal numbers of trading strategies are created for each of the foreign exchange market datasets. To generate a balanced dataset, each training dataset is tailored to a classification system by labelling the training data using that classification system's categorisation metric and value. Half of the trading strategies in the resultant balanced training dataset are above the categorisation metric value and the other half below.

While the training sets are balanced, there is no guarantee that there will be balance under the different market conditions of subsequent market segments. Table 37 shows the number of trading strategies that are below and above each classification system's categorisation metric value on validation and outsample (trading on the datasets in the manner described in Section 8.1.2, but with different metrics for success and failure). The table is coloured from red (lowest quintile) to green (highest quintile) to visualise the imbalances in the number of trading strategies above and below the categorisation metric value. It is noted that these imbalances are similar on both validation and outsample data. The imbalance is less severe when using the trade success rate metric. Creating a balanced training dataset using the expected payoff metric at value -500 resulted in a heavily imbalanced dataset for validation and outsample data; only 175 and 262 trading strategies out of the 10,000 trading strategies fell below the metric value on validation and outsample data respectively.

	Validation		Outsample	
	Below	Above	Below	Above
Expected Payoff -100.0	479	9207	643	9184
Expected Payoff -500.0	175	9337	262	9413
Expected Payoff 0.0	7032	2921	6290	3665
Expected Payoff 100.0	9550	363	9529	408
Expected Payoff 500.0	9778	120	9745	105
Profit Factor 0.75	5264	4683	4581	5364
Profit Factor 0.25	2030	7810	2159	7742
Profit Factor 1.0	7023	2924	6369	3582
Profit Factor 1.5	8768	1181	8180	1788
Profit Factor 2.0	9345	612	8935	1026
Profit Factor 0.5	3944	5958	3843	6077
Return -1000.0	6710	3285	5912	4084
Return -10000.0	866	9130	668	9327
Return 0.0	7059	2893	6239	3710
Return 10000.0	9898	98	9787	209
Return 1000.0	7952	2043	7435	2561
Sharpe Ratio 1.0	9631	275	9281	478
Sharpe Ratio 0.5	9490	449	9277	596
Sharpe Ratio -0.5	1756	8103	1961	7938
Sharpe Ratio 0.0	7166	2781	6271	3691
Sortino Ratio 1.0	8311	1399	8218	1333
Sortino Ratio 0.0	2926	6998	3564	6363
Sortino Ratio -0.5	709	9198	1366	8551
Sortino Ratio 0.5	6174	3695	6071	3740
Trade Success Rate 0.7	7558	2338	7587	2275
Trade Success Rate 0.3	5347	4570	5025	4900
Trade Success Rate 0.5	6552	3229	6437	3342
Trade Success Rate 0.6	7233	2646	7209	2626
Trade Success Rate 0.4	5975	3893	5730	4131

Table 37: Number of trading strategies below and above each classification system's categorisation metric and value in the validation dataset and outsample datasets (coloured with the lowest red to the highest green)

11.3 Classification System Categorisation

Table 38 shows the number of trading strategies that were classified below and above each classification system's categorisation metric value on training, validation and outsample data. The classification systems were obtained from the 200th iteration of the Adaboost algorithm. As in previous chapters, the classification systems use the bootstrap aggregation and feature bagging techniques; they are trained using a balanced multimarket dataset of

10,000 trading strategies and the trading strategies use one technical analysis interpretation. The table cells are coloured from red to green (as previously) indicating imbalances in the numbers of trading strategies above and below the categorisation metric value.

Almost equal numbers of trading strategies were classified above and below each classification system's categorisation metric value on training data. This reflects the numbers of trading strategies above and below the categorisation metric value in the balanced training dataset. Unfortunately, the numbers of trading strategies classified above and below the categorisation metric value on validation and outsample data do not reflect the imbalances evident in Table 37. However, it is also worth noting that the classification systems are not biased towards classifying trading strategies in the dataset as all above or all below the categorisation metric value.

	Training		Validation		Outsample	
	Below	Above	Below	Above	Below	Above
Expected Payoff -100.0	5009	4987	6808	3188	6959	3037
Expected Payoff -500.0	5066	4930	6896	3100	6727	3269
Expected Payoff 0.0	5033	4963	6799	3197	6486	3510
Expected Payoff 100.0	4847	5149	4211	5785	3609	6387
Expected Payoff 500.0	4873	5123	3318	6678	3182	6814
Profit Factor 0.75	5408	4588	7514	2482	6906	3090
Profit Factor 0.25	5210	4786	7188	2808	6709	3287
Profit Factor 1.0	4979	5017	6738	3258	5810	4186
Profit Factor 1.5	4674	5322	6866	3130	4540	5456
Profit Factor 2.0	4758	5238	6910	3086	4286	5710
Profit Factor 0.5	5229	4767	6909	3087	6233	3763
Return -1000.0	5051	4945	6850	3146	7043	2953
Return -10000.0	4788	5208	7617	2379	7709	2287
Return 0.0	5030	4966	6954	3042	6263	3733
Return 10000.0	5272	4724	4826	5170	3211	6785
Return 1000.0	5030	4966	5733	4263	5294	4702
Sharpe Ratio 1.0	4932	5064	3228	6768	3076	6920
Sharpe Ratio 0.5	4895	5101	3462	6534	3346	6650
Sharpe Ratio -0.5	5191	4805	7318	2678	7095	2901
Sharpe Ratio 0.0	4857	5139	6556	3440	5908	4088
Sortino Ratio 1.0	4815	5181	2559	7437	3155	6841
Sortino Ratio 0.0	4978	5018	3210	6786	3840	6156
Sortino Ratio -0.5	5092	4904	2921	7075	5364	4632
Sortino Ratio 0.5	4671	5325	2429	7567	2894	7102
Trade Success Rate 0.7	5223	4773	3543	6453	4202	5794
Trade Success Rate 0.3	5271	4725	4948	5048	5103	4893
Trade Success Rate 0.5	5023	4973	4607	5389	4933	5063
Trade Success Rate 0.6	5178	4818	3586	6410	4312	5684
Trade Success Rate 0.4	5184	4812	4687	5309	5051	4945

Table 38: Number of trading strategies classified as below or above the classification system's categorisation metric value on the training dataset, validation dataset and outsample dataset. The classifiers in the classification system were obtained from the 200th iteration of the Adaboost algorithm (coloured by quintile, with red the lowest and green the highest)

11.4 Analysis of Classification System Results by Performance Measure

Table 39 shows the standard performance measures of each classification system on validation and outsample data. The classifiers were obtained at the 200th iteration of the Adaboost algorithm. The results are again coloured by quintile (red low, green high).

The following sections will discuss the performance of the classification systems in Table 39. Subsequent sections will inspect the classification systems at each iteration of the Adaboost algorithm to locate optimal classification system iterations from results created from validation data.

	Accuracy		Precision		Recall		NPV		Specificity	
	Validation	Outsample	Validation	Outsample	Validation	Outsample	Validation	Outsample	Validation	Outsample
Expected Payoff -100.0	0.37	0.36	1.00	0.99	0.33	0.32	0.07	0.09	1.00	0.96
Expected Payoff -500.0	0.33	0.35	1.00	0.99	0.32	0.33	0.03	0.04	1.00	0.90
Expected Payoff 0.0	0.59	0.58	0.31	0.43	0.34	0.41	0.71	0.67	0.69	0.68
Expected Payoff 100.0	0.45	0.40	0.06	0.06	0.94	0.98	0.99	1.00	0.44	0.38
Expected Payoff 500.0	0.34	0.33	0.02	0.02	0.92	0.97	1.00	1.00	0.34	0.32
Profit Factor 0.75	0.62	0.58	0.68	0.70	0.36	0.40	0.60	0.53	0.85	0.80
Profit Factor 0.25	0.45	0.51	0.94	0.94	0.33	0.40	0.26	0.29	0.92	0.91
Profit Factor 1.0	0.59	0.56	0.33	0.40	0.36	0.47	0.72	0.67	0.69	0.61
Profit Factor 1.5	0.65	0.46	0.13	0.17	0.33	0.53	0.88	0.81	0.69	0.45
Profit Factor 2.0	0.67	0.46	0.07	0.11	0.35	0.63	0.94	0.91	0.69	0.44
Profit Factor 0.5	0.63	0.64	0.88	0.84	0.45	0.52	0.52	0.52	0.90	0.85
Return -1000.0	0.50	0.54	0.23	0.41	0.22	0.30	0.63	0.59	0.64	0.71
Return -10000.0	0.24	0.29	0.82	0.98	0.21	0.24	0.06	0.08	0.51	0.94
Return 0.0	0.59	0.55	0.31	0.40	0.32	0.40	0.72	0.64	0.70	0.64
Return 10000.0	0.49	0.33	0.02	0.02	0.95	0.78	1.00	0.99	0.49	0.32
Return 1000.0	0.56	0.57	0.23	0.31	0.48	0.57	0.81	0.79	0.59	0.57
Sharpe Ratio 1.0	0.35	0.35	0.04	0.07	0.93	0.93	0.99	0.99	0.33	0.32
Sharpe Ratio 0.5	0.39	0.38	0.07	0.08	0.98	0.90	1.00	0.98	0.36	0.35
Sharpe Ratio -0.5	0.43	0.47	0.97	0.96	0.32	0.35	0.23	0.26	0.96	0.95
Sharpe Ratio 0.0	0.57	0.55	0.29	0.41	0.35	0.45	0.72	0.65	0.66	0.61
Sortino Ratio 1.0	0.38	0.44	0.18	0.19	0.95	0.96	0.97	0.98	0.29	0.36
Sortino Ratio 0.0	0.59	0.56	0.71	0.66	0.69	0.64	0.32	0.39	0.34	0.42
Sortino Ratio -0.5	0.68	0.48	0.93	0.87	0.71	0.47	0.08	0.15	0.29	0.57
Sortino Ratio 0.5	0.56	0.60	0.45	0.48	0.93	0.91	0.89	0.88	0.34	0.41
Trade Success Rate 0.7	0.59	0.65	0.36	0.39	1.00	1.00	1.00	1.00	0.46	0.54
Trade Success Rate 0.3	0.85	0.86	0.81	0.86	0.89	0.85	0.90	0.86	0.82	0.87
Trade Success Rate 0.5	0.77	0.80	0.59	0.64	0.98	0.94	0.98	0.96	0.67	0.72
Trade Success Rate 0.6	0.62	0.69	0.41	0.45	1.00	0.98	1.00	0.99	0.49	0.58
Trade Success Rate 0.4	0.81	0.83	0.69	0.75	0.94	0.89	0.95	0.91	0.73	0.79

Table 39: Performance values of each classification system at the 200th iteration of the Adaboost algorithm on validation and outsample data (coloured by quintile, with red the lowest and green the highest)

11.4.1 Trade Success Rate Results

The classification systems with the highest accuracy values used the trade success rate categorisation metric. Before discussing the performance of the classification systems using the trade success rate categorisation metric, it is important to note that it is possible to create profitable trading strategies that make more losing trades than winning trades. For example, a trading strategy with a trade success rate of 0.1 could make nine losing trades out of ten, each of -£10, but make a £150 profit from a winning trade. This trading strategy is expected to average a £6 profit per trade. The trade success rate categorisation metric would label such a trading strategy as 'bad' (whereas a 'good' label would have being given with the return metric used in previous chapters).

The classification system using the trade success rate categorisation metric at value 0.3 obtained the highest accuracy values of 0.85 and 0.86 on validation and outsample data. The classification system's negative predictive value is 0.9 and 0.86 and the specificity is 0.82 and 0.87 on validation and outsample data. This indicates that the classification system is good at classifying trading strategies with a trade success rate less than 0.3 and can identify most of them. The classification system's precision values are 0.81 and 0.86 and the classification system's recall is 0.89 and 0.85 on validation and outsample data. This indicates that the classification system is good at classifying trading strategies with a trade success rate greater than 0.3 and can identify most of them.

The classification systems using the trade success rate categorisation metric at value 0.4 and 0.5 also achieve above random accuracy, precision, recall, negative predictive value and specificity. These classification systems had lower accuracy, precision and specificity performance than that using the trade success rate categorisation metric at value 0.3. These systems, however, obtained higher recall and negative predictive value performance. Similarly, the classification systems using the trade success rate categorisation metric at value 0.6 and 0.7 had lower accuracy, precision and specificity than the systems using a lower categorisation metric value, but achieved higher recall and negative predictive value performance. The classification systems using the trade success rate metric at values 0.6 and 0.7 saw precision and specificity values drop to random or worse than random performance on validation and outsample data.

As the trade success rate metric categorisation value increased, the classification systems achieved better classification performance at classifying trading strategies below the categorisation value and were better at identifying trading strategies above the categorisation value. However as the trade success rate metric categorisation value increased, the classification systems perform worse at classifying trading strategies above the categorisation value and worse at identifying trading strategies below the categorisation value.

The classification system with the trade success rate categorisation metric and categorisation value of 0.3 is particularly good at classifying and identifying the trading strategies above and below the categorisation metric value. Traders with low trade success rate performance are mathematically more likely to experience large losing streaks. As previously mentioned, it is possible for trading strategies to have many small losses but that these losses can be offset by infrequent large profitable trades. If the trader is discouraged by the losing streaks then it is possible that the trader may abandon the strategy before the profitable trade. These classification systems can be deployed to help identify traders that may need additional monitoring as their mental state can affect their ability to execute a trading strategy. These classification systems can also help identify traders that may need extra education to increase their confidence and understanding of their trading style, as these traders are the most likely to experience long losing streaks.

11.4.2 Expected Payoff, Profit Factor and Return Results

Similar to the classification systems used in the previous chapters, the classification systems using the expected payoff, profit factor and return categorisation metrics at values 0, 1 and 0 respectively all attempt to separate the trading strategies into profitable and unprofitable categorisations. These systems achieved negative predictive values of 0.71, 0.72 and 0.72 on validation data and 0.67, 0.67 and 0.65 on outsample data, and also achieved specificity values of 0.69, 0.69 and 0.7 on validation data and 0.68, 0.61 and 0.64 on outsample data. These classification systems were better than random at classifying and identifying unprofitable trading strategies. However, they are worse than random at classifying and identifying profitable trading strategies.

The specificity values of the classification systems using the profit factor categorisation metric with value 0.75 and 0.5 achieve between 0.8 and 0.9 specificity. However, the negative predictive value of these classification systems is random or just above random. These two systems can identify a large proportion of trading strategies with values below the categorisation value, but incorrectly classifies trading strategies that score above the categorisation value. The classification systems using the profit factor categorisation metric at values 0.25, 0.5 and 0.75 have precision values of 0.94, 0.88 and 0.68 on validation data and 0.94, 0.84 and 0.7 on outsample data. These classification systems are good at classifying trading strategies above the categorisation metric, but only 33% to 52% of the trading strategies scoring above the categorisation metric are correctly identified.

None of classification systems using the expected profit, profit factor and return categorisation metrics achieved above random performance on both precision and recall. Those using negative expected payoff categorisation values, profit factor categorisation values less than one and the return categorisation value of -10,000 had positive precision but only obtained recall values of between 0.21 to 0.52.

Classification systems using the expected payoff categorisation value of 100 and 500, and the system using the return categorisation metric value of 10,000 were able to identify most of the trading strategies above their respective categorisation value. The recall values range from 0.92 to 0.98 on validation and outsample data, and the classification system using the return categorisation metric value of 10,000 had a recall of 0.78 on outsample data. Though these classification systems are poor at classifying trading strategies above their categorisation value, they are good at identifying those strategies that exist above the categorisation value.

None of the classification systems using expected profit, profit factor or return categorisation metrics had above random performance for all of the performance values.

11.4.3 Sharpe Ratio and Sortino Ratio Results

Classification systems using the Sharpe ratio categorisation metric with values 1 and 0.5 and the Sortino ratio categorisation metric with value 1 have negative predictive values of 0.97 or above using validation and outsample data; the system using the Sortino ratio categorisation value of 0.5 achieved 0.89 and 0.88 on validation and outsample data. The

specificity of these four classification systems however only ranged between 0.29 and 0.41. These systems are very good at classifying trading strategies that have values below the categorisation value but do not identify the majority of them.

The specificity of the classification system using the Sharpe ratio categorisation metric value of -0.5 is 0.96 and 0.95 on validation and outsample data, indicating that the classification system is good at identifying most of the trading strategies below the categorisation value. However, the negative predictive values of this classification system are only 0.23 and 0.26 on validation and outsample data respectively.

As previously mentioned, classification systems using the Sharpe ratio and Sortino ratio categorisation metrics with values 1 and 0.5 have high negative predictive values on both validation and outsample data. These classification systems also have recall values of 0.9 or above, but have very poor precision performance. These systems are able to identify most of the trading strategies above their respective categorisation values. However, the majority of trading strategies are mislabelled as being above this categorisation metric value. The classification system using the Sortino ratio categorisation metric at value 0.5, however, has a precision value of 0.45 and 0.48 on validation data and outsample data, and so slightly more than a half of the trading strategies are mislabelled above the categorisation metric value.

None of the classification systems using the Sharpe ratio or Sortino ration categorisation metrics had above random performance for any of the performance values. The classification system using the Sharpe ratio categorisation metric at value 0 achieved above random performance on both negative predictive value and specificity for both the validation and outsample datasets. The classification system using the Sortino ratio categorisation metric at value 0 achieved above random performance on both precision and recall for both the validation and outsample datasets.

11.5 Performance at Different Iterations of the Adaboost Algorithm

The following sections analyse each classification system's performance at each iteration of the Adaboost algorithm to attempt to locate an optimal iteration, using validation dataset results, for choosing the classification system to be used on outsample data. Only those

results illustrating the concern of the individual sections are quoted. Full results can be found in Appendix B.

11.5.1 Accuracy

Figures 177 and 178 show the accuracy performance of classification systems using different categorisations on validation and outsample data. Using validation data, the accuracy values of some classification systems increase before converging (on individual values - there is no general convergence), with convergence occurring between the 50th and 100th iteration of the Adaboost algorithm. The accuracy values of the classification systems on outsample data, however, converge more quickly. The validation dataset is used to find optimal classification systems for use on outsample data. No classification systems showed accuracy values that increased to a state of low bias then decrease to a state of high variance. This may be due to bootstrap aggregation and random features technique stopping the classification system from reaching a state of high variance within the 200 iterations of the Adaboost algorithm.

The classification systems categorising trading strategies using the trade success rate metric at values 0.3, 0.4 and 0.5 obtain the best accuracy performance values at the 200th iteration of the Adaboost algorithm, which are 0.77 or above on both validation and outsample data. The accuracy performance converges within a few iterations.

At the 200th iteration of the Adaboost algorithm, the classification system using the return metric categorization metric at value -10,000 obtains the lowest accuracy values of the tested classification systems, with accuracy values of 0.24 and 0.29 on validation and outsample data respectively. Out of the classification systems using the return metric, the classification system with the highest accuracy values categorised trading strategies using the return value of 1,000 pips. At the 200th iteration of the Adaboost algorithm, this classification system achieved an accuracy value of 0.56 and 0.57 on validation and outsample data respectively. The classification system, however, seems to have overfitted on validation data. The classification system's accuracy jumped up and down from 0.63 to 0.56 multiple times before dropping to an accuracy value of 0.56, though this accuracy movement is not observed on outsample data. The accuracy of the classification system

using the return categorisation metric at value -1,000 increases with iteration to accuracy values of 0.5 and 0.54 on validation and outsample data respectively.

At the 200th iteration of the Adaboost algorithm, the highest accuracy classification systems are those using the profit factor metric categorization metric at values of 0.5 and 0.75. These classification systems obtain an accuracy value of 0.63 and 0.62 on validation data and 0.64 and 0.58 for outsample data. At the 200th iteration of the Adaboost algorithm, the classification systems using the profit factor categorisation values of 1.5 and 2 achieve accuracy values of 0.65 and 0.67 on validation data but perform worse than random on outsample data. At the 200th iteration of the Adaboost algorithm, the accuracy values of the classification system using the profit factor value of 1 to categorise trading strategies is 0.59 on validation data but demonstrates near random performance with an accuracy value of 0.56 on outsample data.

The classification systems using the expected payoff metric to categorise trading strategies shows similar accuracy values between the validation dataset and outsample dataset. At the 200th iteration of the Adaboost algorithm, the classification systems using the expected payoff metric value categorization metric of value 0 has an accuracy value of 0.59 and 0.58 on validation and outsample data respectively. The classification systems using the expected payoff metric of values -500, -100, 100 and 500 all have accuracy values worse than random on both validation and outsample data. The classification system using the expected payoff metric value of 100 improves with each iteration and does not converge on validation and outsample data.

The classification systems using the Sharpe ratio categorisation metric also have similar accuracy values on the validation dataset and the outsample dataset, with classification systems using the categorization values of -0.5, 0.5 and 1 showing worse than random performance. At the 200th iteration of the Adaboost algorithm, the classification system using the Sharpe ratio categorisation metric at value 0 have near random accuracy values of 0.57 and 0.55 on validation and outsample data.

At the 200th iteration of the Adaboost algorithm, the accuracy values of the classification systems using the Sortino ratio categorization metric values of 0 and 0.5 are (respectively) 0.59 and 0.56 on validation data and 0.56 and 0.6 on outsample data. The system using the

Sortino categorization metric value of -0.5 at the 200th iteration of the Adaboost algorithm shows an accuracy value of 0.68 on validation data but an accuracy value of only 0.48 on outsample data. For the classification system using the categorisation metric value of 1, at the 200th iteration of the Adaboost algorithm, the accuracy values are worse than random.

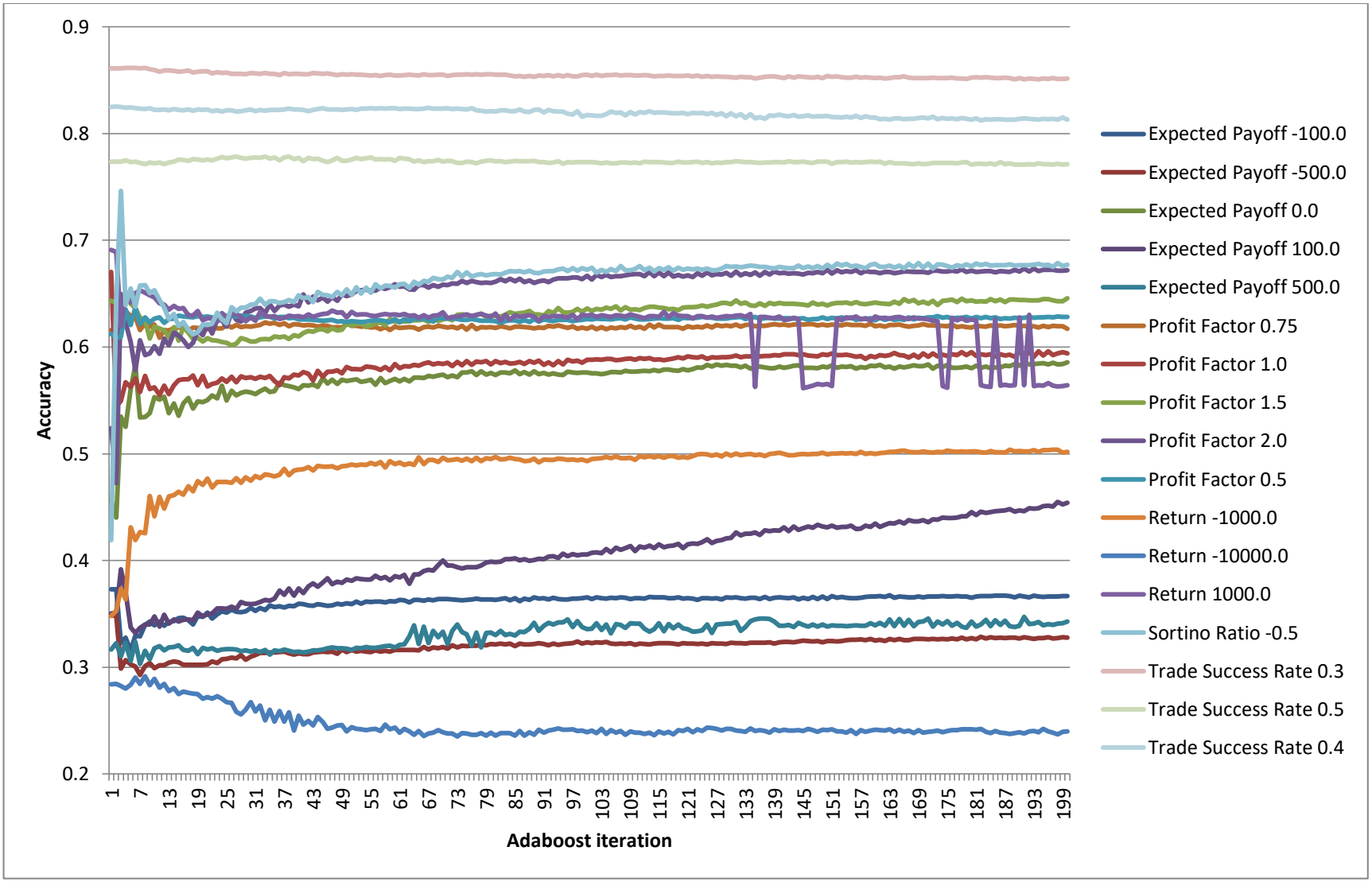


Figure 177: Accuracy of classification systems on validation data using different categorisation metrics and values

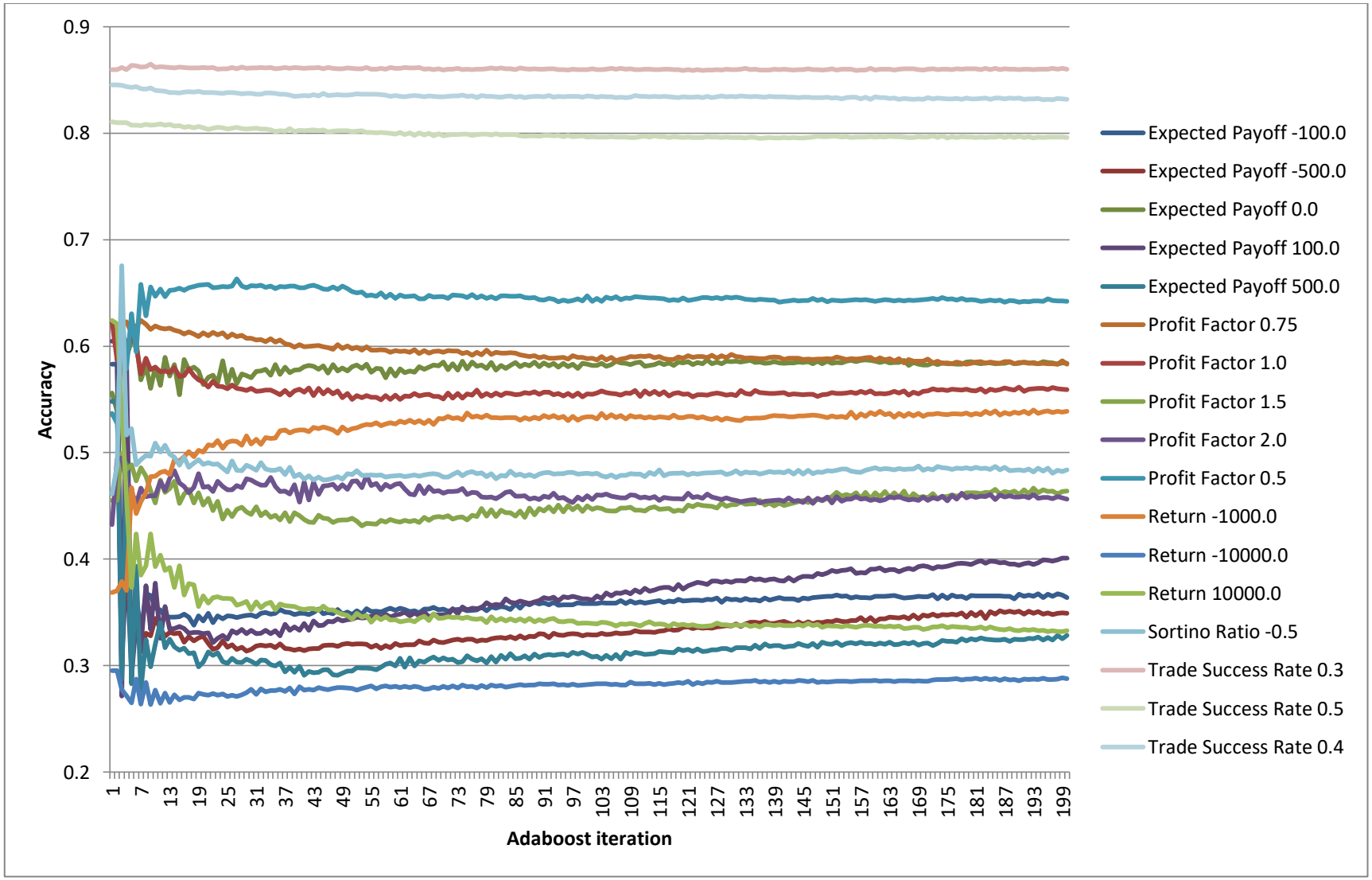


Figure 178: Accuracy of classification systems on outsample data using different categorisation metrics and values

11.5.2 Precision

Figures 179 and 180 show the precision performances of classification systems using different categorisation metrics and values on validation and outsample data. On validation data, the precision values of some classification systems increase before converging before the 50th iteration of the Adaboost algorithm. The precision values of the classification systems on outsample data, however, converge more quickly.

For the trade success rate categorisation metric, precision increases with decreasing metric value. The precision values are worse than random in classification systems using the trade success rate categorisation metric at values 0.6 and 0.7. The systems using the trade success rate categorisation metric at values 0.3, 0.4 and 0.5, however, perform better than random at the 200th iteration of the Adaboost algorithm with precision values of 0.81, 0.69 and 0.59 on validation data and 0.86, 0.75 and 0.64 on outsample data respectively.

The classification systems using the return metric values of -1,000, 0, 1,000 and 10,000 to categorise trading strategies have precision values worse than random. At the 200th iteration of the Adaboost algorithm, the system using the return metric categorisation value of 10,000 performs the worst, with a precision value of 0.02 on both the validation and outsample data. The precision values of these classification systems using the return metric indicates how successful the classification system is at classifying trading strategies above the return metric values. The system using the return categorisation metric value of -10,000 obtains 0.82 and 0.98 precision at the 200th iteration of the Adaboost algorithm. The system using a metric value of 1,000 seems to have overfitted on validation data, as evidenced by the irregular 'jumping' in precision values.

The precision of the classification systems using the profit factor metric values of 1, 1.5 and 2 to categorise trading strategies is worse than random, and as the categorisation value of the profit factor metric increases, the precision decreased. The precision of the systems using the profit factor metric values of 0.25, 0.5 and 0.75, however, have precision values of 0.94, 0.88 and 0.68 on validation data and 0.94, 0.84 and 0.7 on outsample data.

At the 200th iteration of the Adaboost algorithm, the precision values of the classification systems using the expected payoff metric value of 100 and 500 to categorise trading strategies are worse than random for both validation and outsample data. Table 37, above,

showed the number of trading strategies above and below the categorisation value in the validation and outsample dataset. The classification systems using the expected payoff categorisation metric of values 100 and 500 have few trading strategies in the validation and outsample datasets that are above the categorisation value, and this could decrease the precision performance of the classification system. The precision values at the 200th iteration of the Adaboost algorithm are near perfect for the classification systems using the expected payoff categorization metric of values -500 and -100. In contrast, the classification systems using the expected payoff categorisation metric of value -500 and -100 have many trading strategies in the validation and outsample datasets that are above the categorisation value, which could increase the precision performance of the classification system.

Precision values for systems using the Sharpe ratio metric values of 0.5 and 1 show that they are poor at classifying trading strategies above these values, perhaps explained, from Table 37, by there being few trading strategies above the categorisation metric value in the validation and outsample datasets. The classification system using the Sharpe ratio metric value of 0 has worse than random precision, but the system using a metric value of -0.5 achieves precision values of 0.97 and 0.96 on validation and outsample data respectively. From Table 37, 81% and 79% of the trading strategies are above the categorisation metric value on validation and outsample data.

The precision of the classification systems using the Sortino ratio categorization metric values of 0.5 and 1 indicates worse than random performance. The classification system using the Sortino ratio categorization metric value of 0 achieved precision values of 0.71 and 0.66 on validation and outsample data respectively. For a metric value of -0.5, far better precision values of 0.93 and 0.87 on validation and outsample data are achieved. From Table 37, there are 92% and 86% of the trading strategies are above the categorisation value.

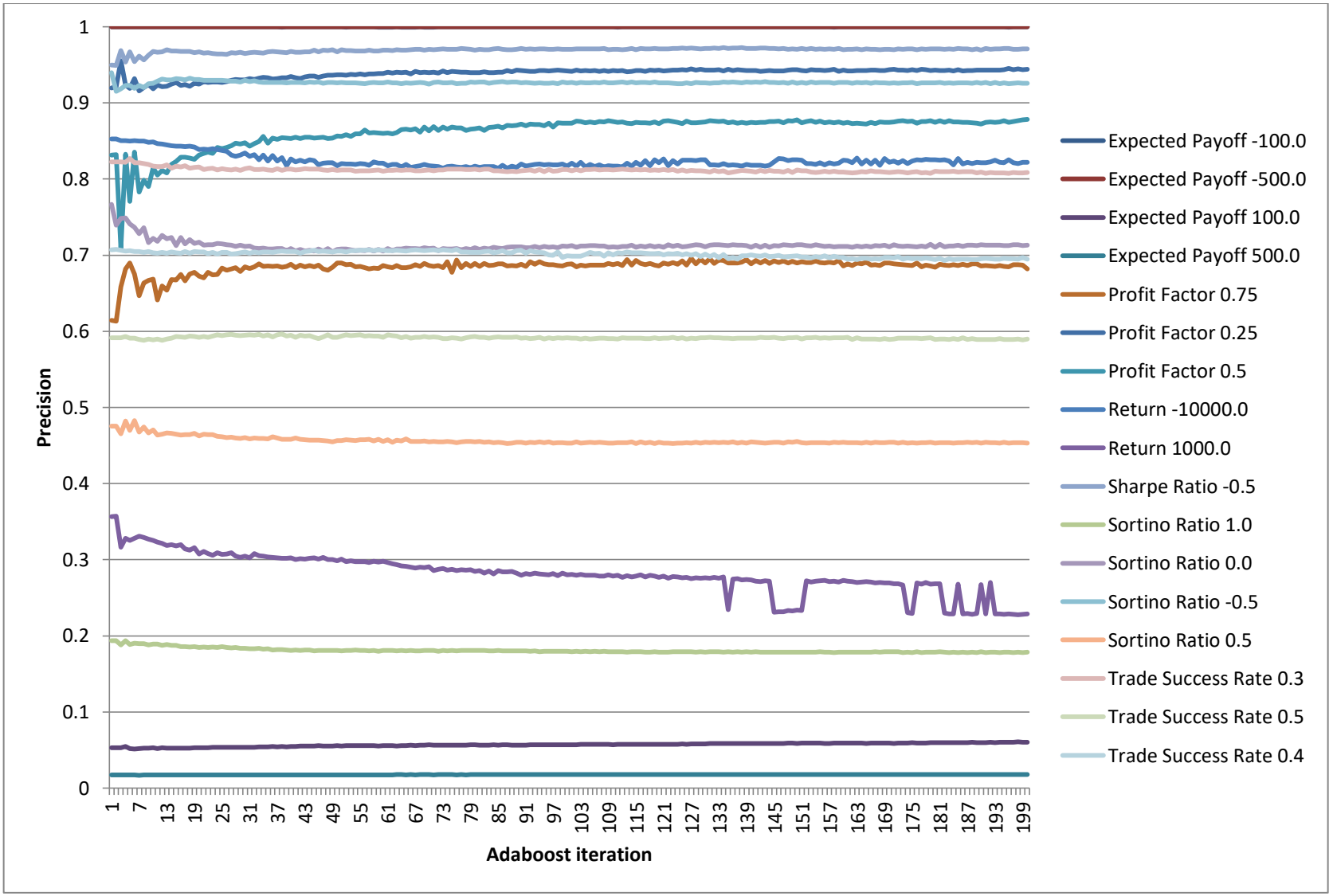


Figure 179: Precision of classification systems on validation data using different categorisation metrics and values

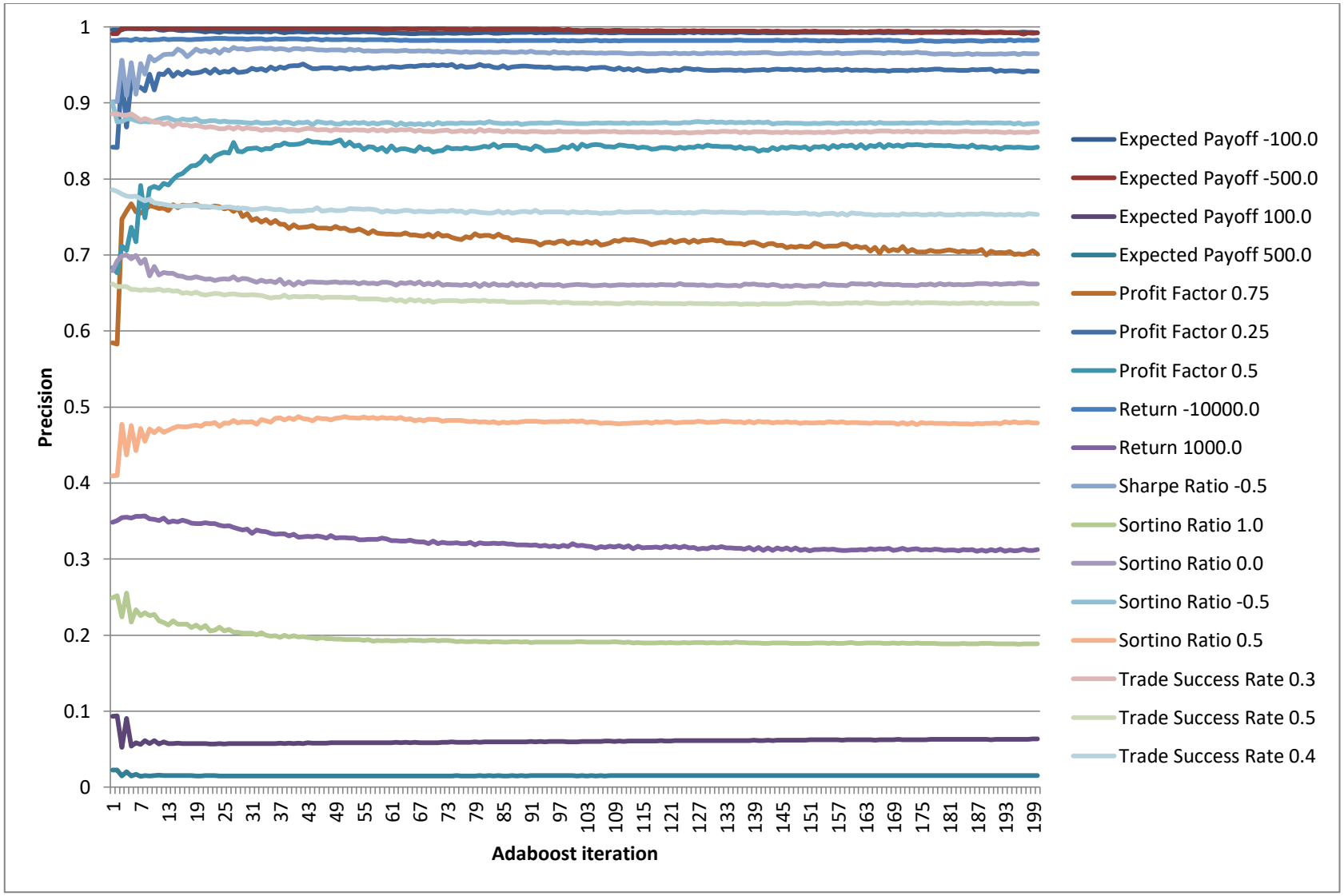


Figure 180: Precision of classification systems on outsample data using different categorisation metrics and values

11.5.3 Recall

Figures 181 and 182 show the recall performance of classification systems using different categorisation metrics and values on validation and outsample data. Using validation data, the recall values of some classification systems increase before converging between the 20th to 100th iteration.. Again, convergence is quicker on outsample data. The recall values of the classification system using the profit factor categorisation metric at value 2 increase around the 13th iteration before decreasing, for both datasets, suggesting this iteration as providing the optimal classification system.

Classification systems using the trade success rate metric at values 0.3, 0.4, 0.5, 0.6 and 0.7 to categorise trading strategies all obtain very high recall values at the 200th iteration of the Adaboost algorithm: 0.89, 0.94, 0.97, 0.998 and 0.997 (respectively) on validation data and 0.85, 0.89, 0.94, 0.98 and 0.995.

The recall values of the classification systems using the return categorisation metric at values -10,000, -1,000 and 0 indicate worse than random performance on both validation and outsample data. The classification system using the return categorisation metric of value 1,000 achieves a random recall performance value of 0.48 on validation, with slightly better than random performance of 0.57 on outsample data at the 200th iteration of the Adaboost algorithm. The recall of the classification system using the return categorization metric at value 10,000 is high, at 0.95 on validation data and 0.78 on outsample data at the 200th iteration of the Adaboost algorithm. Returning to Table 37, the number of trading strategies above the return categorisation metric value of 10,000 is only 98 and 209 out of 10,000 trading strategies in the validation and outsample dataset and the precision of the classification system on validation and outsample data is 0.02. This indicates that the classification system found the majority of trading strategies with a return greater than 10,000 on validation and outsample data. However, the classification system misclassified a lot of trading strategies above the categorization metric value.

The recall values for all classification systems using the profit factor categorisation metric indicate random or worse than random performance at identifying trading strategies above the categorisation value. The only outlier is the classification system using a categorisation metric at value 2, for which an optimal Adaboost iteration can be chosen. On validation

data, the recall of the classification system at each iteration of the Adaboost algorithm increases to 0.74 at the 11th iteration of the Adaboost algorithm before declining to 0.36. On outsample data, this classification system's recall value rises to 0.7 at the 11th iteration before declining to 0.64 at the 200th iteration of the Adaboost algorithm.

The recall values of the classifications systems using the expected payoff metric at values -500, -100 and 0 indicate worse than random performance on both validation and outsample data. In contrast, the systems using the expected payoff categorisation metric at values 100 and 500 obtain recall values of 0.94 and 0.92 (respectively) on validation data and 0.98 and 0.97 on outsample data at the 200th iteration of the Adaboost algorithm. It is noted that for categorisation metric values of 100 and 500, the numbers of trading strategies above the categorisation metric are 363 and 120 (respectively) on validation data and 408 and 105 on outsample data and the precision of the classification system on validation and outsample data is 0.06 and 0.02 (see Tables 37 and 39).

For classification systems using the Sharpe ratio classification metric at values -0.5 and 0, the recall values indicate worse than random performance on validation and outsample data. The recall values of the systems using the Sharpe ratio metric at values 0.5 and 1, however, achieve 0.98 and 0.93 respectively on validation data and 0.9 and 0.93 on outsample data at the 200th iteration of the Adaboost algorithm.

Tables 37 and 39 show the following for the classification systems using the Sharpe ratio categorisation metric value of 0.5 and 1. The number of trading strategies above the categorisation metric are 449 and 275 respectively on validation data, and 596 and 478 on outsample data. The precision of the classification system is 0.07 and 0.04 respectively on validation data and 0.08 and 0.07 on outsample data.

The recall values of the classification systems using the Sortino ratio metric at values 0, 0.5 and 1 to categorise trading strategies all achieve high recall performance of 0.69, 0.93 and 0.95 respectively on validation data and 0.64, 0.91 and 0.96 on outsample data at the 200th iteration of the Adaboost algorithm. For a metric of -0.5, random recall performance is evident on outsample data, but a recall value of 0.71 is achieved on validation data at the 200th iteration of the Adaboost algorithm.

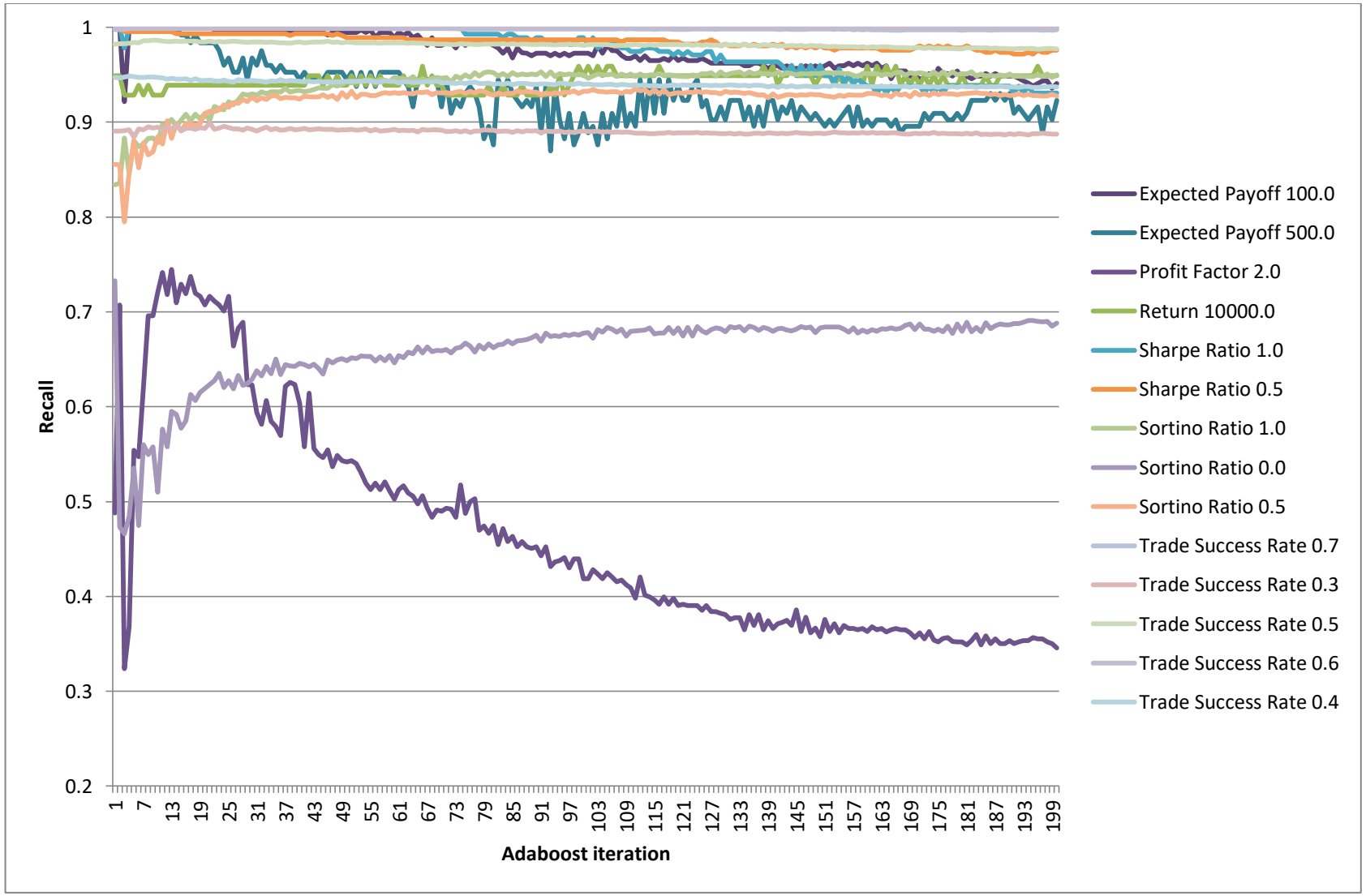


Figure 181: Recall of classification systems on validation data using different categorisation metrics and values

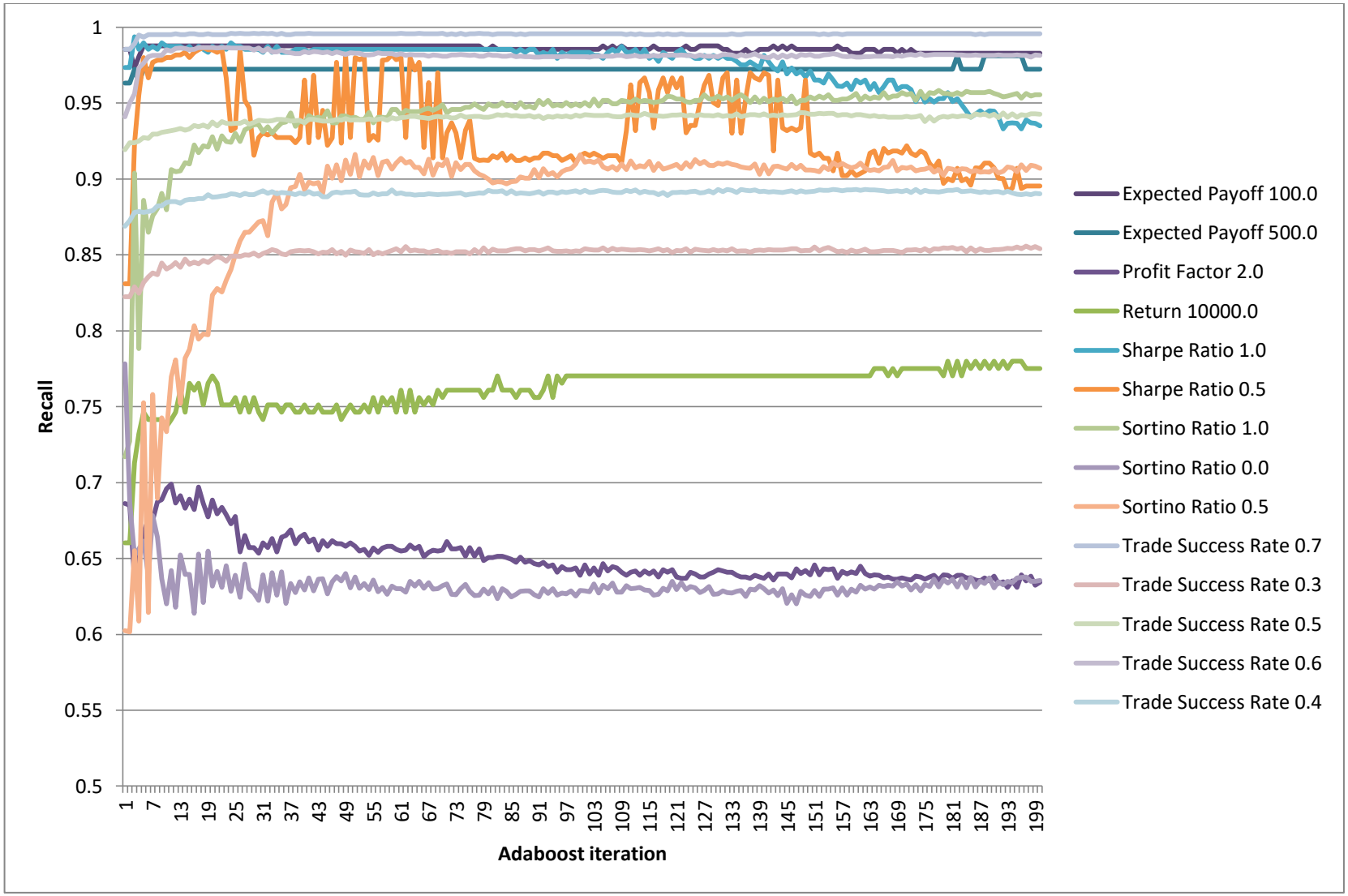


Figure 182: Recall of classification systems on outsample data using different categorisation metrics and values

11.5.4 Negative Predictive Value

Figures 183 and 184 show the negative predictive value performance of classification systems using different categorisation metrics and values on validation and outsample data. When using validation data, the negative predictive values of some classification systems increase within the first 50 iterations, before converging..

The classification systems using the trade success rate metric at values 0.3, 0.4, 0.5, 0.6 and 0.7 to categorise trading strategies all achieve good values for this performance measure at the 200th iteration of the Adaboost algorithm. For the validation dataset, the values are 0.9, 0.95, 0.98, 0.999 and 0.998 respectively. For the outsample dataset, values of 0.86, 0.91, 0.96, 0.99 and 0.998 are achieved for trade success rate metric values 0.3, 0.4, 0.5, 0.6 and 0.7 respectively.

At the 200th iteration of the Adaboost algorithm, the negative predictive value of the classification system using the return categorisation metric at value -10,000 obtains negative predictive values of 0.06 and 0.08 for validation and outsample data respectively. The other classification systems using the categorisation metric at values -1,000, 0, 1,000 and 10,000 achieve performance measure values of 0.63, 0.72, 0.81 and 0.999 on validation data and 0.59, 0.64, 0.79 and 0.985 on outsample data at the 200th iteration of the Adaboost algorithm. The negative predictive value of the classification system using the return categorisation metric at value 10,000 converges at around the 50th iteration using both validation and outsample data.

The negative predictive value of the classification system using the profit factor categorisation metric at value 0.25 performs worse than random, and at value 0.5 the performance is merely random. For a profit factor categorisation metric value of 0.75, a performance measure value of 0.6 is achieved on validation data, but random performance is indicated on outsample data. The negative predictive value of the classification systems using the profit factor metric at 1, 1.5 and 2 indicate good performance, with values of 0.72, 0.88 and 0.94 respectively on validation data and 0.67, 0.81 and 0.91 on outsample data at the 200th iteration of the Adaboost algorithm.

The negative predictive value of the classification system using the expected payoff metric at values -500 and -100 indicates much worse than random performance, while the system

using the expected payoff metric at value 0 achieves a negative predictive value of 0.71 and 0.67 for validation and outsample data respectively at the 200th iteration of the Adaboost algorithm. The systems using the expected payoff metric at value 100 and 500 to categorise trading strategies achieve negative predictive values of 0.99 for both validation data and outsample data at that final iteration.

The system using the Sharpe ratio metric at value -0.5 is indicated to perform worse than random for this performance measure. The system using the Sharpe ratio metric at value 0 achieves a negative predictive value of 0.72 and 0.65 for validation and outsample data respectively at the 200th iteration of the Adaboost algorithm. Sharpe ratio metric values of both 0.5 and 1 have a negative predictive value of 0.99 using validation data. These same Sharpe ratio metric values achieve negative predictive values of 0.99 and 0.98 respectively on outsample data at the 200th iteration of the Adaboost algorithm.

The negative predictive value of the classification system using the Sortino ratio metric at values 0 and -0.5 to categorise trading strategies indicates worse than random performance. The classification systems using the Sortino ratio metric at value 0.5 and 1 to categorise trading strategies achieve negative predictive values of 0.89 and 0.97 respectively on validation data and a negative predictive value of 0.97 and 0.98 on outsample data at the 200th iteration of the Adaboost algorithm.

.

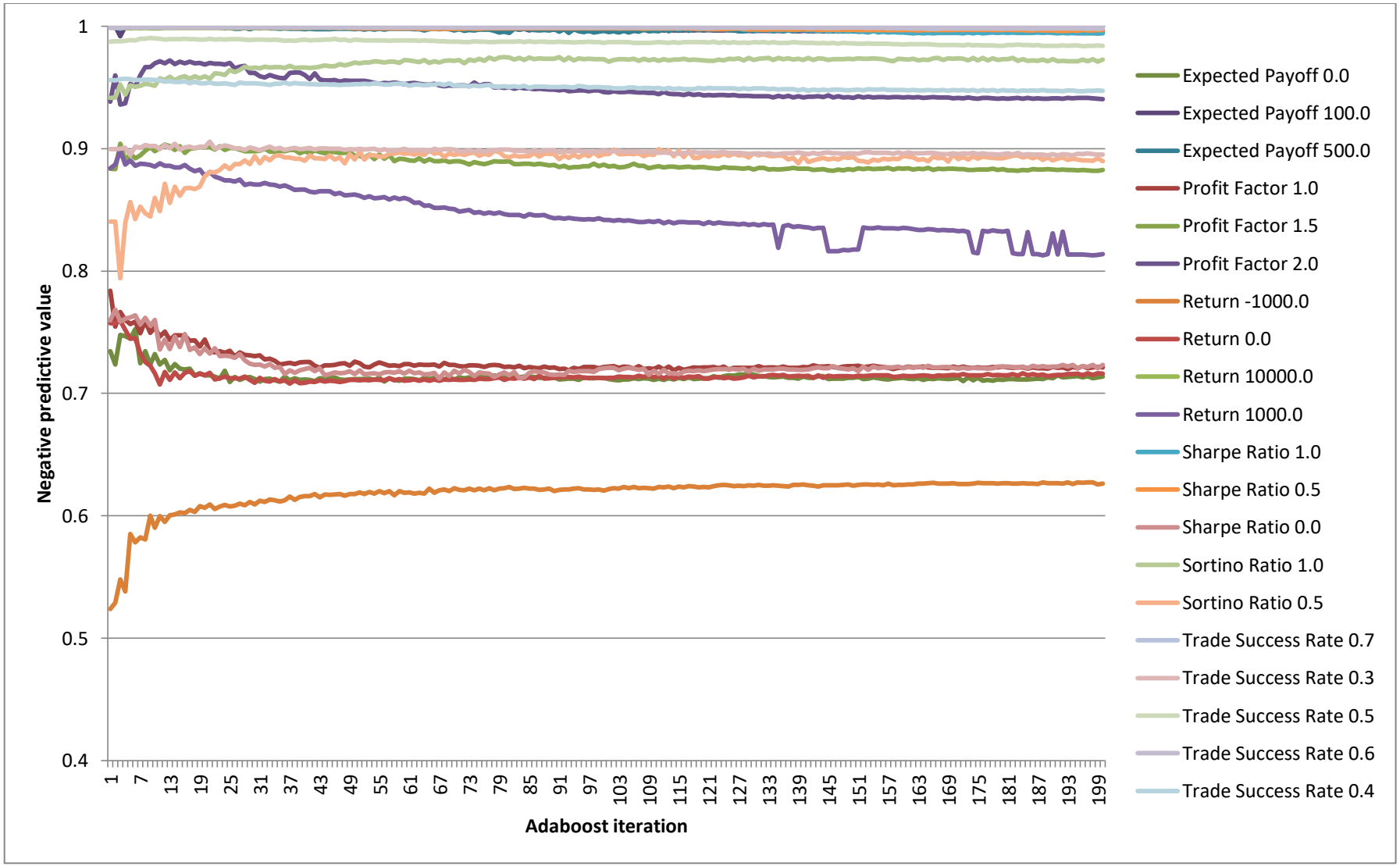


Figure 183: Negative predictive value of classification systems on validation data using different categorisation metrics and values

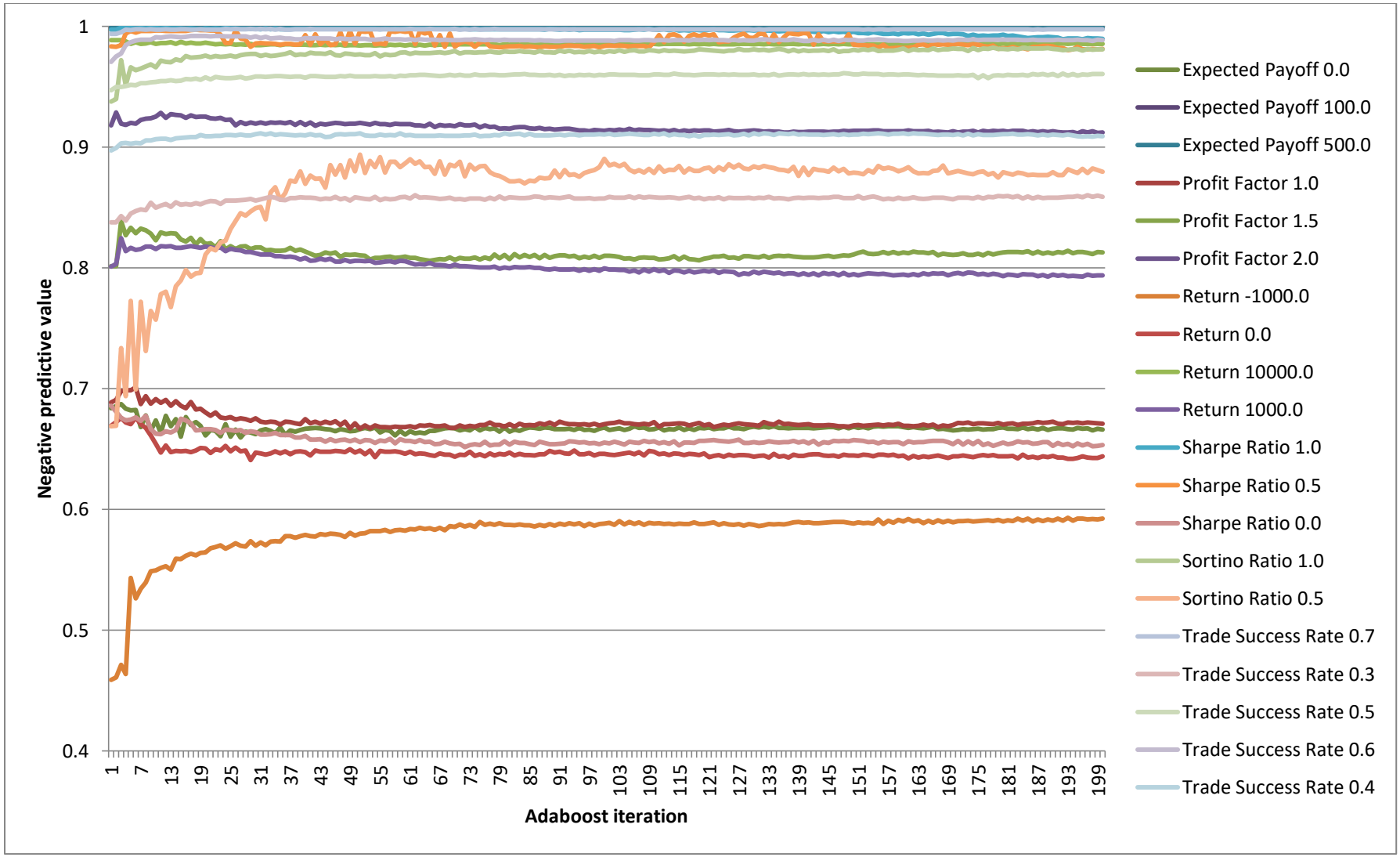


Figure 184: Negative predictive value of classification systems on outsample data using different categorisation metrics and values

11.5.5 Specificity

Figures 185 and 186 show the specificity performance of classification systems using different categorisation metrics and values on validation and outsample data. The specificity values of some classification systems increase, between the 50th to 100th iteration, before converging when using validation data, with faster convergence again evident for outsample data.

The specificity of the classification system using the trade success rate metric at values 0.6 and 0.7 shows slightly worse than random performance on validation data and slightly better than random performance with a specificity value of 0.58 and 0.54 on outsample data. The systems using the trade success rate metric at values 0.3, 0.4 and 0.5 achieve specificity of 0.82, 0.73 and 0.67 respectively on validation data and of 0.87, 0.79 and 0.72 on outsample data at the 200th iteration of the Adaboost algorithm.

The specificity values of the classification systems using the return categorisation metric at values -10,000 and 10,000, at the 200th iteration of the adaboost algorithm, show random performance on validation data. However, the classification systems achieve specificity values of 0.94 and 0.32 respectively on outsample data. For metric values of -1,000, 0 and 1,000, specificity values of 0.64, 0.7 and 0.59 respectively are obtained on validation data and a specificity of 0.71, 0.64 and 0.57 on outsample data at the 200th iteration of the Adaboost algorithm.

The classification systems using the profit factor metric at value 0.25, 0.5 and 0.75 achieve a specificity of 0.92, 0.9 and 0.85 respectively on validation data and of 0.91, 0.85 and 0.8 on outsample data at the 200th iteration of the Adaboost algorithm. A specificity value of 0.69 is achieved for the classification systems using the profit factor metric at both values 1.5 and 2 on validation data, but slightly worse than random performance is observed on outsample data at the 200th iteration of the Adaboost algorithm. The specificity of the classification system using the profit factor categorisation metric at value 1 indicates better than random performance with a specificity value of 0.69 on validation data and a specificity of 0.61 on outsample at the 200th iteration of the Adaboost algorithm. The specificity of the classification system using the profit factor categorisation metric at value 2 changes sharply multiple times, indicating overfitting on validation data, but not on outsample data.

By the specificity measure, the classification system using the expected payoff metric at value 0 performs better than random, with a value of 0.69 on validation data and 0.68 on outsample data at the 200th iteration of the Adaboost algorithm. The classification systems using the expected payoff metric at values 100 and 500 perform worse than random. The specificity of the classification systems using the expected payoff metric at values -500 and -100 have near perfect specificity performance. However, from Tables 35 and 37, 175 and 479 trading strategies were below the categorisation value on validation data and 262 and 643 were below on outsample data. Additionally, the negative predictive value of the these classification systems is less than 0.1 for the validation and outsample dataset.

For the Sharpe ratio metric at values 0.5 and 1, performance by specificity is worse than random. The classification systems using the Sharpe ratio metric at values -0.5 and 0 achieve specificity of 0.96 and 0.66 respectively on validation data, and 0.95 and 0.61 on outsample data at the 200th iteration of the Adaboost algorithm. For metric value at -0.5, specificity is 0.23 on validation data and 0.26 on outsample data. The other classification system, using the Sharpe ratio categorisation metric at value 0, has better than random specificity and negative predictive value performance of 0.72 and 0.65 on validation and outsample data.

The specificity of the classification systems using the Sortino ratio metric at value 0, 0.5 and 1 show worse than random performance. The system using the Sortino ratio metric at value -0.5 to categorise trading strategies achieves a specificity of 0.29 on validation data and a specificity of 0.57 on outsample data at the 200th iteration of the Adaboost algorithm.

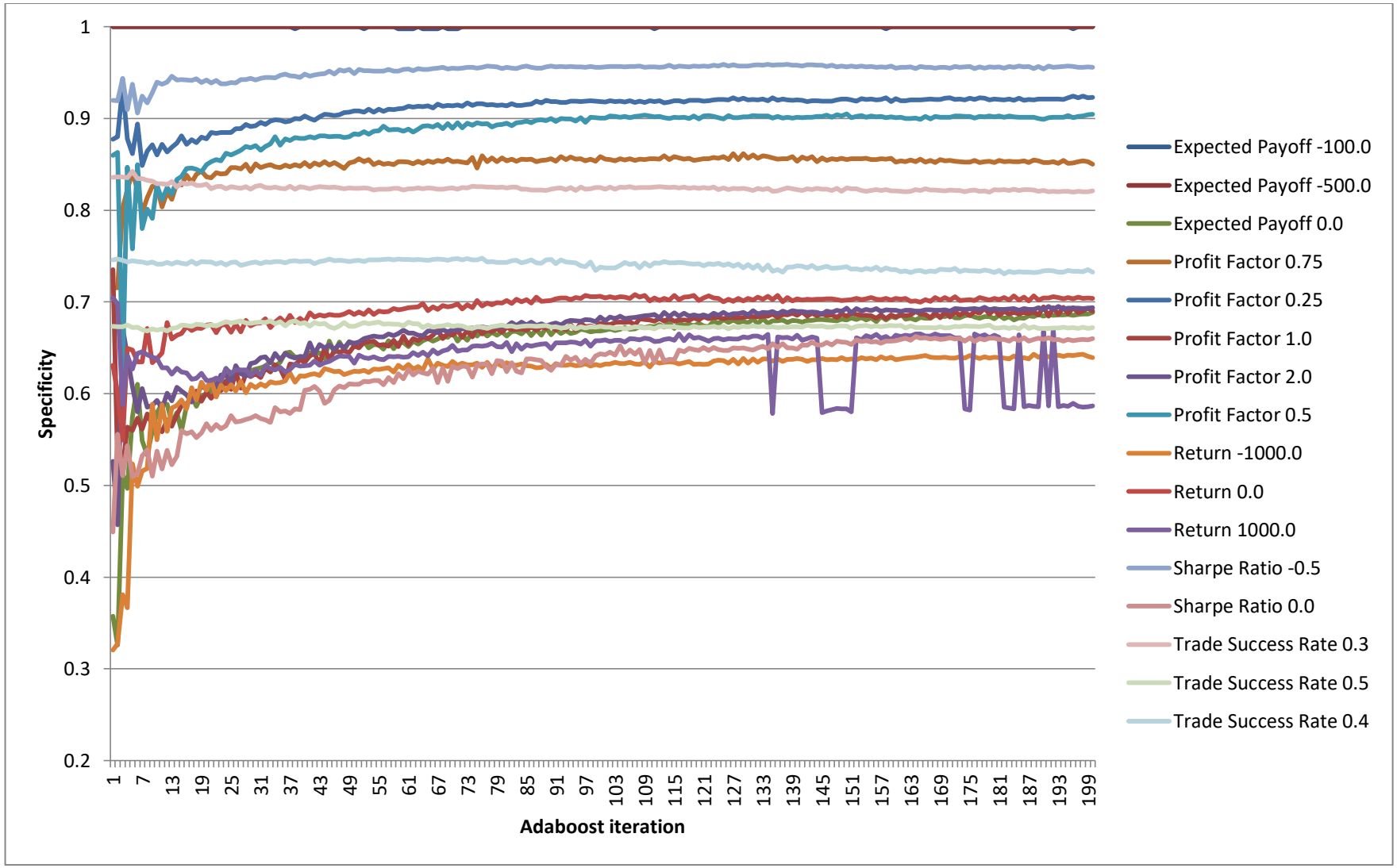


Figure 185: Specificity of classification systems on validation data using different categorisation metrics and values

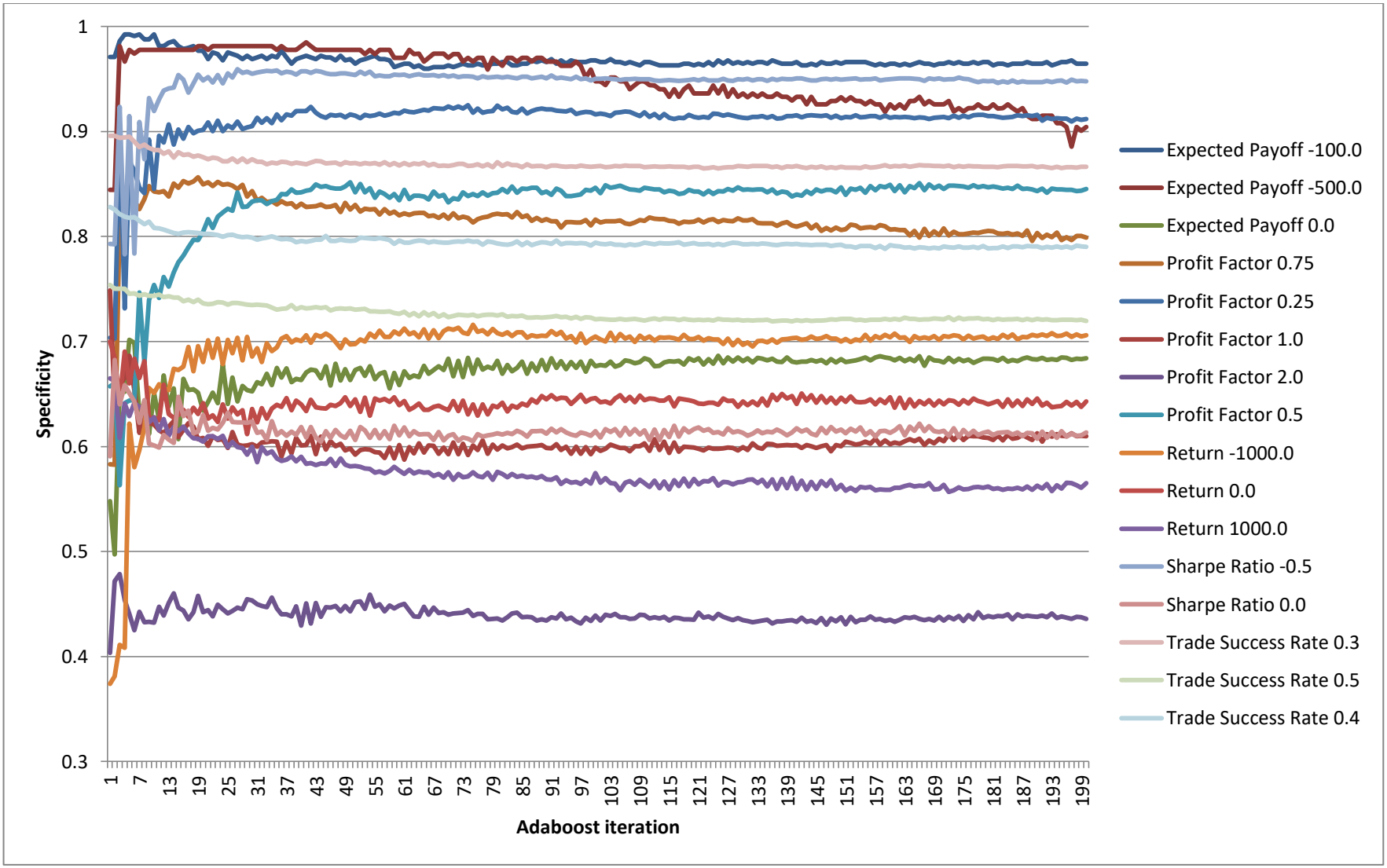


Figure 186: Specificity of classification systems on outsample data using different categorisation metrics and values

11.6 Summary

It was previously demonstrated that classification systems using a balanced multimarket training dataset, the return categorisation metric at value 0, and the bootstrap aggregation and feature bagging technique, was promising for the detection of bad trading strategies. This chapter explored the development of systems that use different classification metrics and values. Each of the trading strategies used to derive the performance metrics contain a single technical analysis interpretation, this size having proved the most consistently effective in previous experiments. As before, the experiments in this chapter also showed that having a balanced training dataset (containing equal numbers of trading strategies above and below each classification system's respective categorisation metric value) offers no assurance of similar balance in the validation and outsample datasets, due to changing market conditions in the consecutive market segments.

Of the tested categorisation metrics and values, the classification systems using the trade success rate categorisation metric experienced least imbalance. In contrast, classification systems that used the return categorisation metric at 10,000 and the expected payoff categorisation metric value of 500 had less than 2% of the 10,000 trading strategies in the dataset above the categorisation metric.

Naturally trading strategies that are above the tested categorisation metrics are seen to be more desirable than trading strategies that are below the tested categorisation metrics.

The higher the categorisation metric value the fewer the number of trading strategies above the categorisation metric value on validation and outsample data, and the lower the categorisation metric value the fewer the number of trading strategies below the categorisation metric value on validation and outsample data.

The performances of most of the tested classification systems were similar on validation and outsample data, with convergence evident in some systems within 50 to 100 iterations on both validation data and outsample results. The choice for the optimal iteration for these classification systems, therefore, would be between the 50^{th} and the 100^{th} iterations as the classification system has low bias and low variance between these iterations. Further iterations would risk overfitting of the classification system, though the bootstrap aggregation and feature bagging techniques would be expected to mitigate against this.

The primary aim of this thesis is create a classification system that can classify and identify 'bad' traders. To achieve this aim, classification systems that have above random negative predictive value and specificity performance on outsample data is needed. The classification systems proposed in this chapter that obtain above random performance in both performance measures are those using the following:

- The expected payoff categorisation metric at value 0
- The profit factor categorisation metric at value 1
- The return categorisation metric at values -1,000, 0 and 1,000
- The Sharpe ratio categorisation metric at value 0

Classification systems that have high specificity performance but poor or random negative predictive value (between 0.2 and 0.55) can also be useful as decision support systems that narrow the pool of traders that are potentially bad. The classification systems proposed in this chapter that employ these performance measures are those using:

- The profit factor categorisation metric at values 0.25, 0.5 and 0.75
- The Sharpe ratio categorisation metric at value -0.5

Classification systems using the trade success rate categorisation metric at values 0.3, 0.4 and 0.5 showed above random accuracy, precision, recall, negative predictive value and specificity performance. As pointed out above, traders with low trade success rate performance are mathematically more likely to experience large losing streaks. They are also more likely to be discouraged by these, and to abandon a strategy that might be in a longer term be successful by reason of infrequent large gains. These classification systems help identify such traders, and to target programs of education and encouragement.

11.7 Visualising Adaboost's Decision Criterion

With the methodology established, and suggestions made for viable classification systems and associated performance metrics, this section explores how Adaboost's decision criteria can be visualised in the proposed system. This section first formalises the model of a trader with respect to the Adaboost formula. The effectiveness of using Adaboost as a tool for improving a trader's understanding of performance metrics is then explored. Finally, graphic illustrations of the relative importance of different performance metrics in respect of the Adaboost derived classifiers are given. These are intended to clarify and further explain the benefit of the work of this thesis for the partner company, and for trading companies more generally.

11.7.1 Trader Decision Making Process

Traders use general 'rules of thumb' (heuristics) to help predict whether a trading strategy will be profitable on future market data. These heuristics are based upon performance metrics derived from historical and/or simulated data. For example, traders seek trading strategies with profit factors above a predetermined threshold, and drawdowns which do not exceed a specified magnitude. Whether formulated explicitly or not, traders effectively assign a weight to each performance metric to indicate its relative importance. Implicitly, traders are employing a scoring mechanism to predict whether a trading strategy is likely to be profitable on future market data, based on the trader's past experience, bias and knowledge. Formulating this explicitly, the scoring mechanism for a trading strategy is here taken to be of the following form,

$$R = \propto_1 h_1 + \propto_2 h_2 + \cdots + \propto_n h_n$$

where, *h* is the performance metric heuristic, α is the weight of importance, and *R* is the trading strategy score.

Each performance metric heuristic h_i outputs the value one (1), if the trading strategy satisfies the heuristic and negative one (-1) if the trading strategy does not satisfy the heuristic. The weight of importance $\propto_i$ for each heuristic is a positive real number.

Traders seek trading strategies that maximise this function *R* to increase the likelihood of future profits. The function *R* can be thought of as a set of criteria which, if maximised, generates the trader's optimal trading strategy.

Trading strategies that are found by maximising only one performance metric, such as the profit factor, may produce trading strategies that have large drawdowns or have undesirable Sharpe ratios, etc. It is essential, but particularly hard to find, the correct trade-off between the different performance metrics given a set of current market conditions. A trader's decision-making criteria, used to judge whether a trading strategy will be profitable in the future, needs to possess this optimal trade-off between performance metric heuristics.

11.7.2 Using Adaboost as a Regression Function

As described in the Section 2.4.3.2, Adaboost produces a weighted function of the form,

$$H = sign(\propto_1 h_1 + \propto_2 h_2 + \cdots + \propto_n h_n)$$

where *h* is the weak classifier value, *α* is the weight, *H* is the strong classifier value, and the 'sign' function indicates whether the summation is positive or negative. The goal of Adaboost is to create a strong classifier, *H*.

The $sign$ function provides the classification threshold. If H is positive then the trading strategy is good, if H is negative then the trading strategy is bad. Notice that Adaboost's strong classifier contains the exact scoring mechanism that a trader was taken to employ in the previous section,

$$H = sign(R)$$

By removing the $sign$ function from Adaboost's strong classifier, the result is a regression function that intended to behave similarly to the way a trader scores trading strategies. In addition to the advantages stated previously, through this regression function Adaboost can effectively automate the work of the trader using a substantial number of performance metrics, trading strategies and market data.

11.7.3 Visualising Adaboost

Figures 187, 188 and 189 show “user-friendly” visualisations intended to help managers and traders understand how important each performance metric is in the classifier’s decision making process. Figure 187 shows the importance of each performance metric on the AUDUSD market using 51 Adaboost classifiers using the bootstrap aggregation and random features technique. The chart reports the average, minimum, maximum, the lower standard deviation of the average, and upper standard deviation of the average of proportion of weight assigned to each performance metric. The performance metrics on the *y* axis are sorted from highest to the lowest average value. The results show that the largest consecutive trade loss, number of unsuccessful trades and average trade loss performance metrics are on average the most important. The results also show the standard deviation of trade loss performance metric is has the highest minimum proportion of weight across all 51 Adaboost classifiers. Figure 188 displays the average importance of each performance metric on each of the 7 foreign exchange markets and reports the average importance. Figure 189 reports the same average importance of each performance metric in pie chart form.

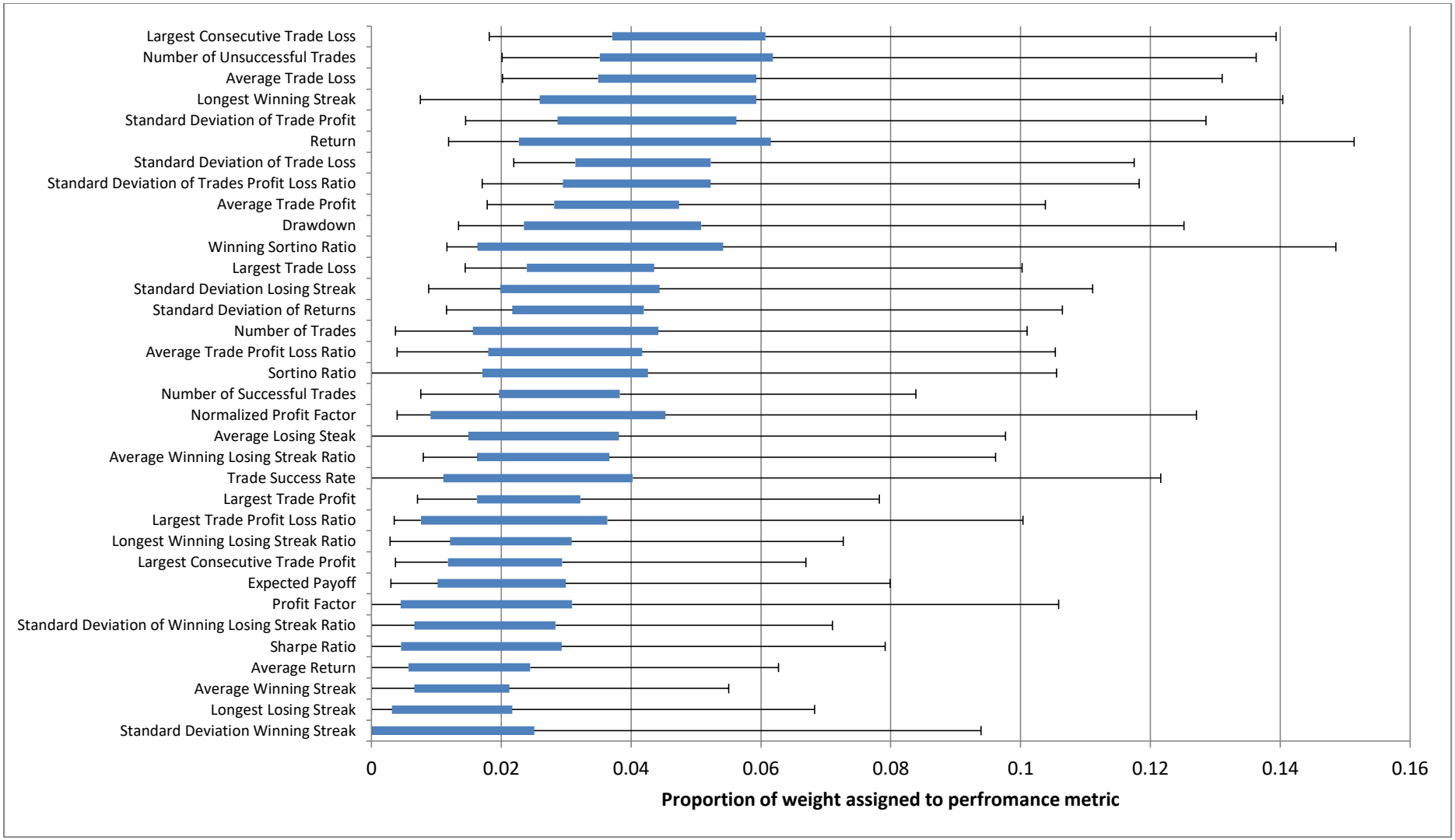


Figure 187: Summary of the importance of each metric for classifying trading strategies in the AUDUSD market

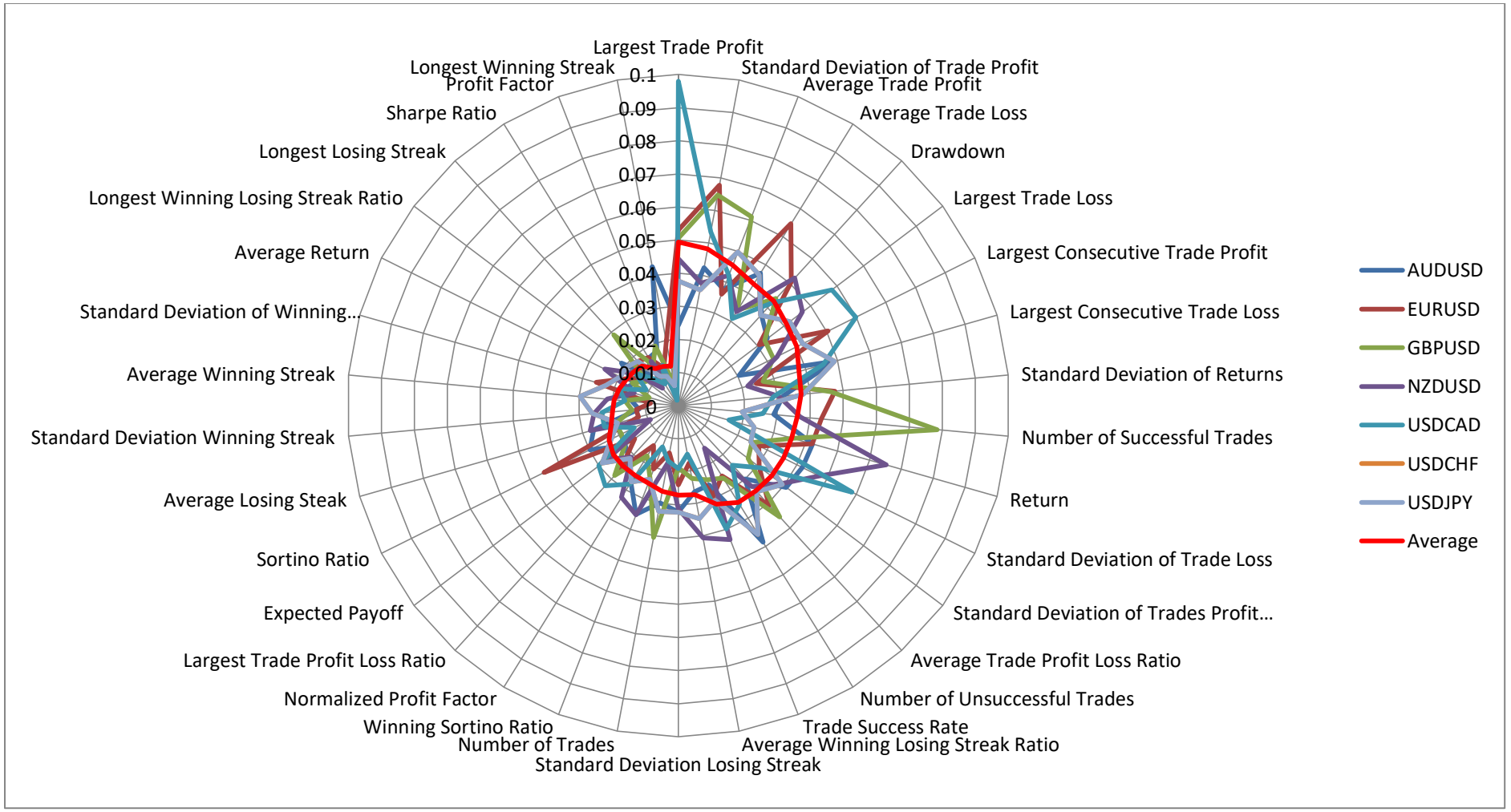


Figure 188: Summary of the average importance of each metric across the 7 forex markets

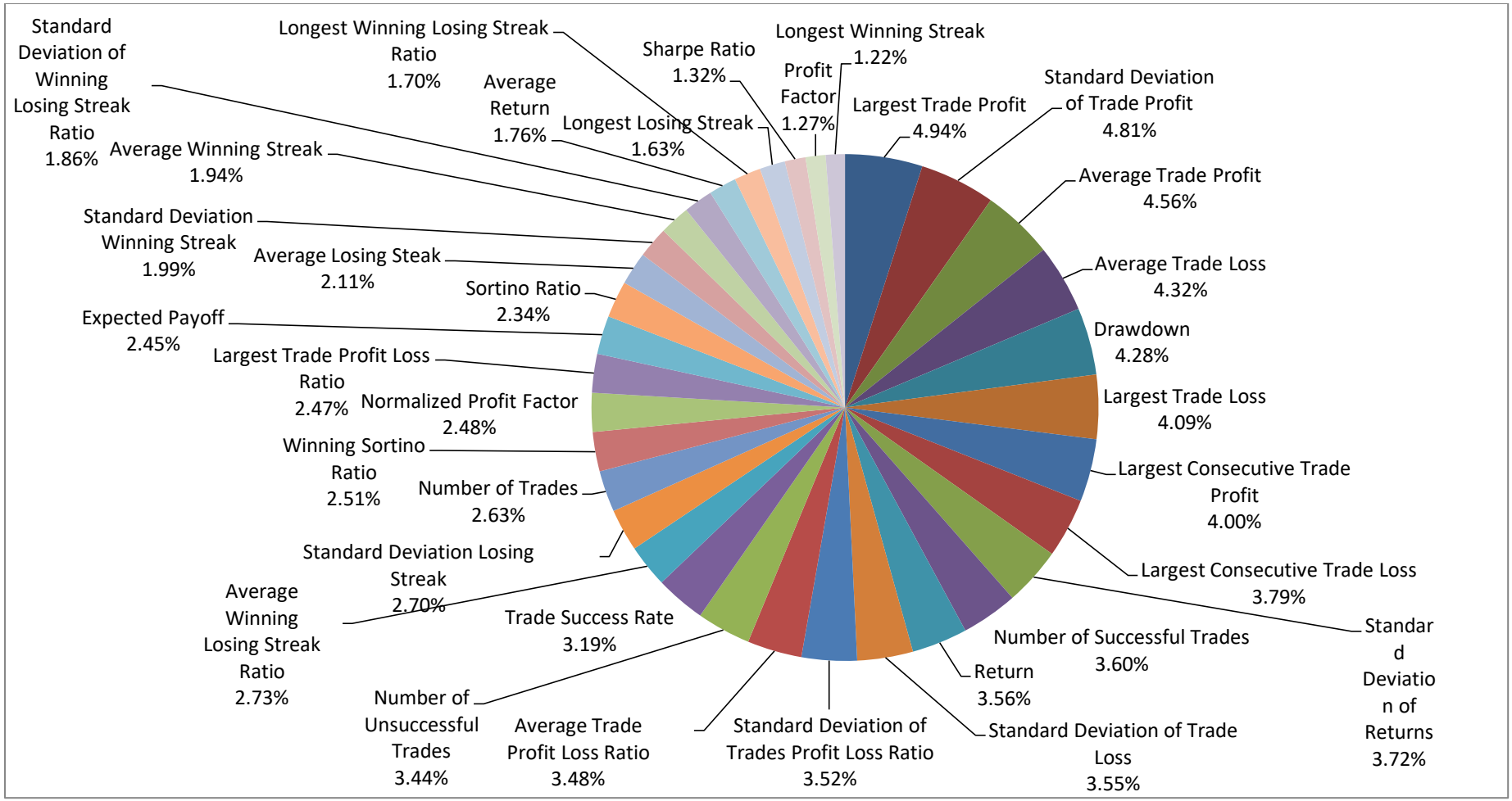


Figure 189: Summary of the average importance of each metric across the 7 forex markets

Managers can use these visualisations to give themselves a greater intuition regarding, and hence more confidence in, the classification system's decision-making process. In particular, the manager can see that the classification system considers a variety of different performance metrics and their weights. Further work might consider only the weak classifiers of a single performance metric in the classification system to find the optimal value for the performance metric.

From these visualisations and further investigation, managers could advise individual traders in improving a set of performance metrics that would increase their score. For newer traders, managers could target areas in which the trader is under-performing to increase their score to an acceptable level.

Chapter 12 – Conclusions and Future Work

This chapter explains how the aims and objectives of this project, presented in Section 1.4, have been met, and highlights the contributions of the work to trading studies more generally (Section 12.1). While the development was conducted in accordance with the specific needs of the project partner, it is also discussed how the outcomes have broader applicability within the sector. Attention is drawn to the potential of the methodology as a topic for further development and as a framework for application to other functions. These themes are elaborated in the section Future Work (Section 12.2).

12.1 Conclusions

The primary aim of the project described in this thesis was to *'construct a decision support system for the classification of traders and thereby the early detection of bad traders'* (Section 1.4).

In developing a methodology for the detection of bad traders, it was first established in Chapter 5 that sector-standard performance metrics obtained from trading strategies traded on historical data can be correlated to future values. This correlation is based on such a large dataset that it can be taken to be significant. This suggests limited but practical predictive applications for the metrics, and in particular for the intended decision support system of this project. The experiments presented in Chapter 8 suggested the suitability of *Adaboost* as a machine learning technique for *learning classifiers* that can distinguish between good and bad trading strategies; the *bootstrap aggregation* and *feature bagging* techniques were also shown to be effective in reducing overfitting. Of the various measures employed to evaluate binary classifiers (described in Section 2.5), *negative predictive value* and *specificity* were also suggested as the most important in meeting the project aims, as they are particularly effective in the classification and identification of bad traders. These first experiments in the use of Adaboost also highlighted the difficulty in identifying an optimal iteration for choosing the eventual classification system. The effectiveness of learning is dependent on the training dataset, specifically the extent to which this is representative of trading strategies. A training dataset that is greatly imbalanced between numbers of good and bad trading strategies was shown to impact adversely on learning in Chapter 9, with the worst results occurring for trading strategies of larger sizes (where the performance of larger sized trading strategies is less reliable generally). That chapter details experiments that demonstrated the benefits of

using *balanced training datasets*. The effectiveness of learning is also adversely impacted upon by changes in market conditions.

The classification systems developed up to this point were all trained on individual foreign exchange markets. Chapter 10 employed training datasets comprised of all 7 markets considered in this thesis to create more general classifiers. The classifiers created on these *multimarket* training datasets were shown to improve on previous results, and trading strategies of size 1 were shown to be the most reliably classified. All of the above experiments employed the return performance metric, but experiments in Chapter 11 with different metrics established the usefulness of *trade success rate* (with a metric value of *0.3*) for the classification of potentially high-risk trading strategies – and through that the early detection of poor trading decision-making. The italicised terms define the proposed and implemented decision support system for the classification of bad trading strategies, and through this the identification of poor traders. (It is repeated here from Chapter 11, however, that the question of the choice of performance metric is still open.)

The principle findings (see Chapter 11) may be summarised as follows.

Distribution of trading strategies

It was observed that the higher the categorisation metric value the fewer the number of trading strategies above this on validation and outsample data, and the lower the categorisation metric value the fewer the number of trading strategies below this on validation and outsample data.

Optimal iteration for choosing classification systems

The choice for the optimal iteration for these classification systems was found to be between the 50^{th} and the 100^{th} iterations in that the classification system has both low bias and low variance between these. Further iterations would risk overfitting of the classification system, though the bootstrap aggregation and feature bagging techniques would be expected to mitigate this.

Effective classification system settings

In pursuing this thesis' primary aim, classification systems having above random negative predictive value and specificity performance were those employing:

- The expected payoff categorisation metric at value 0
- The profit factor categorisation metric at value 1
- The return categorisation metric at values -1,000, 0 and 1,000
- The Sharpe ratio categorisation metric at value 0

Classification systems that have high specificity performance but poor or random negative predictive value (between 0.2 and 0.55) can also be useful as decision support systems that narrow the pool of traders that are potentially bad. Such classification systems were:

- The profit factor categorisation metric at values 0.25, 0.5 and 0.75
- The Sharpe ratio categorisation metric at value -0.5

Novelty of proposed methodology

It was reported in Chapter 2 that, to the best of the current author's knowledge, no proposal has been made in the literature to use Adaboost for the classification of trading strategies or traders. Hence the proposed methodology, and the implementation and analysis of this, constitute the most significant novel contribution of this thesis.

The pragmatic approach taken to the development of the decision support system is rooted in quantitative financial trading logic familiar to those in the financial trading sector. This focus has typically been lacking in academic work to date, and results in the developed system having greater immediate value to the project partner. Specifically, the work employs technical analysis algorithms and performance metrics typically employed by traders and therefore understood by their managers. While these techniques have been used in AI-based forecasting tools previously, their use in a system for classifying trading strategies is novel.

Datasets and rigour of testing

It is also noted that the results presented have been based on large datasets and comprehensive testing which is less evident in the literature for market forecast systems (the only extant work of relevance to this project). For each classification system, a total of 10,000 technical analysis interpretations were used, and for experiments that used trading strategies, 10,000 such strategies were also created. In total, 36 features (performance metrics) were used in the training datasets to create the classification systems, and 33 technical analysis interpretations were used from each of the 7 foreign exchange markets (AUDUSD, EURUSD, GBPUD, NZDUSD, USDCAD, USDCHF and USDJPY). Lastly, for experiments that used multiple markets, all 7 foreign exchange market datasets were used.

Additional findings arising from analysis

The analysis of the results of the experiments performed has contributed several interesting findings. The first of these related to the potential for changes in market conditions across the contiguous market segments used for training, validation and outsample datasets. The proposal in Section 8.1.2 was to train the classification system on the training dataset, and then apply it to the validation dataset in order to determine the optimal iteration for selecting the eventual classifier to be used on outsample data. The experiments highlighted the difficulty in selecting an optimal iteration; the evidence presented is against the existence of a single optimal iteration. It is suggested here that it may be more appropriate to skip the validation stage and simply select the classifier from the 100^{th} iteration of the Adaboost algorithm in training. Chapter 10 also highlighted an improved performance of that classifier on validation data as opposed to outsample, which perhaps might be expected as market conditions change progressively over time – a classifier is more likely to be useful on the very next market segment.

Secondly, the term 'bad' is loosely defined, and must be understood in the context of the relationship between the performance metric and the datasets on which the classifier is trained and used. The approach taken in the methodology presupposes that a single metric has been well-chosen for the judgement that a trade is 'bad' in the given context. Chapter 11

speculated that classification systems using different classification metrics could be built in combination, to create more general systems. This is examined in the passage on Future Work below.

Thirdly, while results are promising for the reliable detection of consistently poor trading strategies (those that are poor under many market conditions), distinguishing between potentially good and consistently good strategies more difficult. Consistently good traders can be taken to employ more and better strategies which apply under a broad range of market conditions. Potentially good traders may employ a similar number of comparably sound strategies, which are not currently returning well due to market conditions; these are likely to improve with experience. It takes considerable data to distinguish apparently good traders from these; such traders are in fact using fewer and/or less sound strategies which happen to be returning better than random results due to statistical reasons. However, bad traders will certainly return no better than random results in the long term. Moreover, such results are likely to be inconsistent, ending even if similar market conditions continue or return.

Last, while Section 5.2.1 suggested that a trading strategy comprising multiple technical analysis interpretations might constitute a good model for a trader, the above methodology was more successful with strategies of size 1. Smaller trading strategies are more likely to behave similarly on future data, with larger sized strategies being potentially too complex to be effective in modelling market behaviour. It is noted here, however, that many traders do employ simple predictive market models, and so the model of a trader employed within the methodology is still realistic.

Evaluation of the usefulness of Genetic Algorithms

The secondary aim of the project was to *'generate trading strategies, as models of traders, by combining the buy and sell signals from technical analysis interpretations'*.

Due to the commercial sensitivity of the project partner's actual trader history data, and the need for a large volume of trade histories, it was decided that a pool of trading strategies (as models of traders) would need to be created artificially. The initial plan for the generation of

this pool of trading strategies was to create chromosomes for use in the Genetic Algorithm, which represented trading strategies as combinations of 3 or more technical analysis interpretations combined through majority vote. The initial pool was created randomly, and then the Genetic Algorithm forced to evolve more effective strategies through increasing population fitness (Chapters 6 and 7). This pool was to be subsequently used to train the classification systems. However, the Genetic Algorithm approach was not successful in improving fitness by use of the return metric (Chapter 6), nor by alternative standard metrics (Chapter 7). It was considered likely that single metrics were not suitable for locating optima within their search spaces, but it was not clear how multi-objective fitness functions could be built. The novel approach taken to modelling traders within the chromosomes may also have contributed to the observed failure. Nevertheless, this work prompted consideration of the Adaboost algorithm as a mechanism for creating a multi-objective fitness function through training on labelled data.

The approach taken as an alternative to generating the pool was to create random trading strategies, of different sizes and balanced between 'good' and 'bad' in relation to a given performance metric, and to develop classification systems trained on these using the Adaboost methodology described above.

Section 1.4 also detailed two objectives for the work. The first objective was to *'identify a set of criteria by which the success or otherwise of a trading strategy may be judged, i.e. the trader who fails to meet these criteria, or a subset of these criteria, can be considered a bad trader.'* This addresses the issue of the definition of 'bad' mentioned above. The experiments that established the methodology for the decision support system identified such a set of criteria, as described above.

Ranking of trading strategies

The second objective was to *'score trading strategies so that they can be ranked and compared to other trading strategies. The propriety trading company employing the traders can therefore make better decisions about the amount of funds to allocate to each trader relative to other traders.'* The project partner is very keen on the use of a white box approach

to categorise traders, as they wish to understand how the classification system makes its decisions. It is difficult to decipher the decision-making processes of artificial neural networks and support vector machines, as these are black boxed approaches to classification and regression. Adaboost, in contrast, is a white boxed approach. The training process results in a classification system which constitutes a weighted sum of contributions from many weak classifiers (which are themselves performance metrics relying on thresholds, *i.e.* on outputs above or below a specified value). The regression version of the Adaboost formula (explained in Section 11.7.1) makes explicit this weighted sum. It is interesting to note that this formula is, in effect, a multi-objective fitness function that could be employed by local search algorithms; this is considered in future work, below. Chapter 11 concludes with a brief discussion on how the ranking of trading strategies enabled through Adaboost could contribute to decision-making by a trading company on the allocation of funds to traders. Classification of a trader as 'bad' would provide justification to reduce those funds allocated to the trader, hence reducing risk to capital, while classification of a trader as 'good' might encourage the allocation of more capital to increase return.

Broader value of the work

While the contributions described above have been framed primarily in terms of their value to the project partner, other trading companies would benefit similarly. Moreover, the project outcomes could be extended to other practical applications.

Within the finance sector, the algorithm can identify weaknesses in strategy that will inform training programmes. The approach taken enables a trading company to access and engage with the classification mechanism. The weighted sum of contributions from many weak classifiers output by Adaboost can provide managers and traders with insights into the trade-offs between the different performance metrics.

More generally, the approach to employing Adaboost for classification could be applied to any sector that generates data on which decisions must be made, and in which poor decisions must be avoided. Investments in the public sector comprise several varied strategies, with measurable outcomes such as pass rates in schools, recovery rates in

hospitals, changes in employment figures *etc*. A weighting amongst alternative strategies would be valuable in determining those with the poorest return. In the private sector, underperforming factories, retail outlets, or entertainment franchises could be identified in terms of their strategies of personnel allocation and resource consumption.

12.2 Future Work

To extend the work presented in this thesis, more experiments can be performed to investigate whether classification systems using performance metrics derived from a different timeframe than the hourly timeframe used might prove better for classifying and identifying 'bad' traders; daily, 5-minute and 1-minute market data are possibilities. Changing the length and dates of the market segments may also yield interesting results. In regard to datasets, unfortunately, OSTC were unable to provide trade histories of their traders (due to reasons of sensitivity and confidentiality). These might have constituted a better training dataset for the classification systems described in this thesis, although it is noted that the volume of data might have been insufficient.

It was suggested in the concluding remarks of Chapter 11 that a final classification system could be created from a combination of multiple classification systems that use different performance metrics and boundaries. This final classification system could be constructed using the Adaboost algorithm or another machine learning classification or regression algorithm which could make sense of the different component classification systems.

Trading strategies in this thesis are, in essence, forecasting the future as they are technical analysis interpretations that attempt to model some continuing or recurrent aspect of the financial market. The model of trading strategies can be extended to include *take profit* levels which trigger the end of a trade when the asset has achieved a particular price or profit, and conversely to include *stop losses* which limit losses if the asset goes against the trading strategy's position (as described in Section 5.1). Further mechanisms for risk management such as time limits on trades could also be investigated.

Some traders deploy a data driven process for the creation of trading strategies and trade thousands of trading strategies in parallel. The creation of thousands of trading strategies in a data-driven approach, are likely to by chance have a proportion of trading strategies that pass the trader's criterion, filters and tests that indicate that the trading strategy is not 'bad'

and should be traded. Classification systems for determining whether a trader is 'good' or 'bad' (which is itself a data-driven model), could be adapted to act as an additional filter for rejecting trading strategies.

In this thesis the classification systems were created using the Adaboost algorithm. The project partner, OSTC, prefer such a white box approach to classification in which the decisions of the classification system can be inspected. Further work on the visualisation and inspection of the Adaboost-based classification systems in this thesis could be explored. Going beyond the preference of the project partner, comparisons between the Adaboost classification systems of this thesis with black box classification systems, such as Artificial Neural Networks and Support Vector Machines, would be interesting. Hybrid classification systems of these types might also produce better classification performance.

Genetic Algorithms have been used to create formulae that act as trading strategies and have been used by Lipinski (2010; 2010; 2004) to evolve a population of trading strategies that are reported to be profitable on future market data. While Lipinski's work modelled trading strategies as a binary chromosome, the representation of a trading strategy in this thesis was a treelike structure. Future work would involve attempting to duplicating Lipinski's work, and then changing the representation of a trading strategy to determine whether controlling the number of technical analysis interpretations per trading strategy can overcome overfitting.

The methodology proposed results in a formula which is, in effect, a multi-objective fitness function; this function is produced by ignoring the 'sign' function that turns the Adaboost regression into a classification. Other local search algorithms such as *Simulated Annealing* and *Tabu search*, and the Genetic Algorithm, could be constructed to employ that multi-objective fitness to explore the search space of trading strategies. This might prove particularly interesting in the case of the Genetic Algorithm, returning to the work of Chapters 6 and 7 in generating pools of artificial trading strategies.

References

Allen, F., & Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. *Journal of Financial Economics, 51*(2), 245-271.

Aranha, C., & Hitoshi, I. (2009). The memetic tree-based genetic algorithm and its application to portfolio optimization. *Memetic Computing*, 139-151.

Aronson, D. (2011). *Evidence-Based Technical Analysis: Applying the Scientific Method and Statistical Interence to Trading Signals.* John Wiley & Sons.

Arth, C., Graz, Limberger, F., & Bischof, H. (2007). Real-Time License Plate Recognition on an Embedded DSP-Platform. *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on* (pp. 1-8). Minneapolis, MN, USA: IEEE.

Asadi, S., Hadavandi, E., Mehmanpazir, F., & Nakhostin, M. M. (2012). Hybridization of evolutionary Levenberg–Marquardt neural networks and data pre-processing for stock market prediction. *Knowledge-Based Systems, 35*, 245-258.

Atsalakis, G., & Valavanis, K. (2009). Surveying stock market forecasting techniques – Part II: Soft computing methods. *Expert Systems with Applications, 36*(3), 5932-5941.

Azoff, M. (1994). *Neural Network Time Series Forecasting of Financial Markets.* New york: John Wiley & Sons, Inc.

Bahlmann, C., Haasdonk, B., & Burkhardt, H. (2002). Online handwriting recognition with support vector machines - a kernel approach. *Frontiers in Handwriting Recognition, 2002. Proceedings. Eighth International Workshop on* (pp. 49-54). IEEE.

Barbe, B. M., & Odean, T. (2000). Trading Is Hazardous to Your Wealth: The Common Stock Investment Performance of Individual Investors. *The Journal of Finance, 55*(2), 773-806.

Barinova, O., & Gavrishchaka, V. V. (2009). Generic Regularization of Boosting-Based Algorithms for the Discovery of Regime-Independent Portfolio Strategies from High-Noise Time Series. *Innovative Computing, Information and Control (ICICIC), 2009 Fourth International Conference on* (pp. 1456-1459). Kaohsiung: IEEE.

Bartram, S. M., & Grinblatt, M. (2017). Agnostic Fundamental Analysis Works. *Journal of Financial Economics (JFE), Forthcoming*, [Epub. ahead of print, available at SSRN: https://ssrn.com/abstract=2479817, last accessed 31/12/2017].

BBC. (2014, September 22). *Tesco suspends execs as inquiry launched into profit overstatement*. Retrieved February 10, 2016, from BBC news: http://www.bbc.com/news/business-29306444

Bell, S. (2016). Quantitative Finance for Dummies. Chichester: John Wiley & Sons, Ltd.

Bergstra, J., Casagrande, N., Erhan, D., Eck, D., & Kégl, B. (2006). Aggregate features and AdaBoot for music classification. *Machine Learning, 65*(2-3), 473-484.

Berutich, J. M., López, F., Luna, F., & Quintana, D. (2016). Robust technical trading strategies using GP for algorithmic portfolio selection. *Expert Systems with Applications, 46*, 307-315.

Bewick, V., Cheek, L., & Ball, J. (2004). Statistics review 13: receiver operating characteristic curves. *Critical Care, 8*(6), 508.

Black, F., & Scholes, M. (1973). The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 637-654.

Bollen, J., Mao, H., & Zeng, X. (2011). Twitter mood predicts the stock market. *Journal of Computational Science, 2*(1), 1-8.

Bollinger, J. (2001). *Bollinger on Bollinger bands.* McGraw Hill.

Booth, A., Gerding, E., & Mcgroarty, F. (2014). Automated trading with performance weighted random forests and seasonality. *Expert Systems with Applications, 41*(8), 3651-3661.

Braun, H. (1990). On solving travelling salesman problems by genetic algorithm. *International Conference on Parallel Problem Solving from Nature. 496*, pp. 129-133. Springer-Verlag Berlin Heidelberg.

Chang, P. C., Liu, C. H., Lin, J. L., Fan, C. Y., & Ng, C. (2009). A neural network with a case based dynamic window for stock trading prediction. *Expert Systems with Applications, 36*(3), 6893.

Chaudhuri, T. D., & Ghosh, I. (2016). Artificial Neural Network and Time Series Modeling Based Approach to Forecasting the Exchange Rate in a Multivariate Framework. *Journal of Insurance and Financial Management, 1*(5), 92-123.

Cheh, J. J., Weinberg, R. S., & Yook, K. C. (2011). An application of an artificial neural network investment system to predict takeover targets. *Journal of Applied Business Research, 15*(4), 33-46.

Chen, A. S., Leung, M., & Daouk, H. (2003). Application of neural networks to an emerging financial market: forecasting and trading the Taiwan Stock Index. *Operation Research in Emerging Economics, 30*(6), 901-923.

Cheng, B. Y., Lee, J. S., & Guo, J. I. (2015). An AdaBoost object detection design for heterogeneous computing with OpenCL. *Consumer Electronics, 2015 IEEE International Conference on* (pp. 286-287). Taipei: IEEE.

Chiu, D. Y., & Chen, P. J. (2009). Dynamically exploring internal mechanism of stock market by fuzzy-based support vector machines with high dimension input space and genetic algorithm. *Expert Systems with Applications, 36*(2), 1240-1248.

Choudhury, S., Ghosh, S., Bhattacharya, A., Fernandes, K. J., & Tiwari, M. K. (2014). A real time clustering and SVM based price-volatility prediction for optimal trading strategy. *Neurocomputing, 131*, 419-426.

Corp, M. S. (n.d.). *MetaQuotes Software*. Retrieved September 10, 2017, from www.metaquotes.net

Cox, J. C., Ross, S. A., & Rubinstein, M. (1979). Option pricing: A simplified approach. *Journal of Financial Economics, 7*(3), 229-263.

Creamer, G., & Freund, Y. (2010). Automated trading with boosting and expert weighting. *Quantitative Finance, 10*(4), 401-420.

Cristianini, N., & Taylor, J. S. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods.* Cambridge: Cambridge Press.

Das, P., & Banerjee, I. (2011). An hybrid detection system of control chart patterns using cascaded SVM and neural network–based detector. *Neural Computing and Applications, 20*(2), 287-296.

Dash, R., & Dash, P. K. (2016). A hybrid stock trading framework integrating technical analysis with machine learning techniques. *The Journal of Finance and Data Science, 2*(1), 42-57.

Demiriz, A., Bennett, K. P., & Shawe Taylor, J. (2002). Linear Programming Boosting via Column Generation. *Machine Learning, 46*(1), 225-254.

Diale, M., Walt, C. V., Celik, T., & Modupe, A. (2016). Feature selection and support vector machine hyper-parameter optimisation for spam detection. *In Pattern Recognition Association of South Africa and Robotics and Mechatronics International Conference* (pp. 1-7). IEEE.

Dietterich, T. G. (2000). Ensemble Methods in Machine Learning. *Lecture Notes in Computer Science, 1857*, 1-15.

Eberhart, R., & Shi, Y. (2011). Computational intelligence - Concepts to implementations. In R. Eberhart, & Y. Shi. Burlington, MA: Morgan Kaufmann Publishers.

Elden, A. S., Malaka A. Moustafa, H. M., & Emara, A. H. (2013). AdaBoost ensemble with simple genetic algorithm for student prediction model. *International Journal for Computer Science and Information Technology, 5*(2), 73-85.

Evans, C., Pappas, K., & Xhafa, F. (2013). Utilizing artificial neural networks and genetic algorithms to build an algo-trading model for intra-day foreign exchange speculation. *Mathematical and Computer Modelling, 58*(5), 1249-1266.

Fariaa, E. L., Albuquerquea, M. P., Gonzalez, J. L., Cavalcantea, J. T., & Albuquerquea, M. P. (2009). Predicting the Brazilian stock market through neural networks and adaptive

exponential smoothing methods. *Expert Systems with Applications, 36*(10), 12506–12509.

Fernández, A., & Gómez, S. (2007). Portfolio selection using neural networks. *Computers and Operations Research, 34*(4), 1177-1191.

Fernandez-Lozano, C., Fernández-Blanco, E., Kirtan Dave, b. N., Gestal, M., Dorado, J., & Munteanu, C. R. (2014). Improving enzyme regulatory protein classification by means of SVM-RFE feature selection. *Molecular Biosystems, 10*(5), 1063-1071.

Freund, Y. (2001). An Adaptive Version of the Boost by Majority Algorithm. *Machine Learning, 43*(3), 293-318.

Freund, Y., & Schapire, R. E. (1995). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Lecture Notes in Computer Science, 904*, 119-139.

Gavrishchaka, V. V. (2006). Boosting-based framework for portfolio strategy discovery and optimization. *New Mathematics and Natural Computation, 2*(3), 315-330.

Gavrishchaka, V. V. (2006). Boosting-Based Frameworks in Financial Modeling: Application to Symbolic Volatility Forecasting. In T. B. Fomby, & D. Terrell, *Econometric Analysis of Financial and Economic Time Series* (pp. 123-151). Emerald Group Publishing Limited.

Ghahramani, Z. (2004). Unsupervised learning. In *Advanced Lectures on Machine Learning* (pp. 72-112). Springer Berlin Heidelberg.

Gupta, P., Mehlawat, M. K., Inuiguchi, M., & Chandra, S. (2014). Multi-criteria Portfolio Optimization Using Support Vector Machines and Genetic Algorithms. In *Fuzzy Portfolio Optimization* (pp. 283-309). Springer Berlin Heidelberg.

Hadavandi, E., Ghanbari, A., & Shavandi, H. (2010). Integration of genetic fuzzy systems and artificial neural networks for stock price forecasting. *Knowledge-Based Systems, 23*(8), 800-808.

Hanley, J. A., & McNeil, B. J. (1983). A method of comparing the areas under receiver operating characteristic curves derived from the same cases. *Radiology, 148*(3), 839-843.

Hao, C., Wang, J., Xu, W., & Xiao, Y. (2013). Prediction-Based Portfolio Selection Model Using Support Vector Machines. *Business Intelligence and Financial Engineering, 2013 Sixth International Conference on* (pp. 567-571). Hangzhou: IEEE.

Hiep, T. K., Duc, N. M., & Trung, B. Q. (2016). Local search approach for the pairwise constrained clustering problem. *In Proceedings of the Seventh Symposium on Information and Communication Technology* (pp. 115-122). ACM.

Hieronymus, T. A. (1977). *Economics of futures trading for commercial and personal profit.* New York: Commodity Research Burea, inc.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems.* Ann Arbor, MI: University of Michigan Press.

Hu, Y., Liu, K., Zhang, X., Su, L., Ngai, E. W., & Liu, M. (2015). Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review. *Applied Soft Computing, 36*, 534-551.

Jackson, W. G., Özcan, E., & John, R. I. (2017). Tuning a Simulated Annealing metaheuristic for cross-domain search. *Evolutionary Computation (CEC), 2017 IEEE Congress on* (pp. 1055-1062). IEEE.

Jordan, S. J. (2014). Is momentum a self-fulfilling prophecy? *Quantitative Finance, 14*(7), 737-748.

Kao, L. J., Chiu, C. C., Lu, C. J., & Yang, J. L. (2013). Integration of nonlinear independent component analysis and support vector regression for stock price forecasting. *Neurocomputing, 99*, 534-542.

Kazem, A., Sharifi, E., Hussain, F. K., Saberi, M., & Hussain, O. K. (2013). Support vector regression with chaos-based firefly algorithm for stock market price forecasting. *Applied Soft Computing, 13*(2), 947-958.

Khalifehloo, Habibi, M., Mohammad, M., & Heydari, M. (2017). Application of artificial neural network and regression analysis to recovery of missing hydrological data in Klang

River Basin. *Environmental Conservation, Clean Water, Air & Soil* (p. 67). Beijing, China: The International Water Asscoiation Publishing.

Khanifar, Hossein, Jamshidi, N., & Mohammadinejad, M. (2012). Studying Affecting Factors on Analysts' Decisions Regarding Share Analysis in Tehran Stock Exchange: A Fundamental Analysis Approach. *European Journal of Economics, Finance and Administrative Sciences, 44*, 77-86.

Kirilenko, A., Kyle, A. S., Samadi, M., & Tuzun, T. (2017). The Flash Crash: High-Frequency Trading in an Electronic Market. *The Journal of Finance, 72*, 967-998.

Kirkpatrick, C. D., & Dahlquist, J. R. (2010). *Technical Analysis: The Complete Resource for Financial Market Technicians.* New Jersey: FT Press.

Kothari, S. P. (2001). Capital markets research in accounting. *Journal of Accounting and Economics, 31*(1), 105-231.

Kryzanowski, L., Galler, M., & Wright, D. W. (1993). Using Artificial Neural Networks to Pick Stocks. *Financial Analysts Journal, 49*(4), 21-27.

Krzysztof, M., Filipiak, P., & Lipiński, P. (2012). Evolutionary Approach to Multiobjective Optimization of Portfolios That Reflect the Behaviour of Investment Funds. *International Conference on Artificial Intelligence: Methodology, Systems, and Applications* (pp. 202-211). Springer, Berlin, Heidelberg.

Kuo, R. J., Chen, C. H., & Hwang, Y. C. (2001). An intelligent stock trading decision support system through integration of genetic algorithm based fuzzy neural network and artificial neural network. *Fuzzy Sets and Systems, 118*(1), 21-45.

Lakshman, R. N., Ramesh, D., Manjula, B., & Govardhan, A. (2012). Prediction of Stock Market Index Using Genetic Algorithm. *Computer Engineering and Intelligent Systems, 3*(7), 162-171.

Lane, G. C. (1985). Lane's Stochastics: the ultimate oscillator. *Journal of Technical Analysis, 21*, 37-42.

Li, C., Chu, X., Chen, Y., & Xing, L. (2015). A knowledge-based initialization technique of genetic algorithm for the travelling salesman problem. *Natural Computation (ICNC), 2015 11th International Conference on* (pp. 188-193). IEEE.

Lim, M. A. (2015). *The Handbook of Technical Analysis+ Test Bank: The Practitioner's Comprehensive Guide to Technical Analysis.* John Wiley & Sons.

Lin, Y., Song, Y., Li, Y., Wang, F., & He, K. (2017). Multilingual corpus construction based on printed and handwritten character separation. *Multimedia Tools and Applications, 76*(3), 4123-4139.

Lipinski, P. (2007). Discovering stock market trading rules using multi-layer perceptrons. *Computational and Ambient Intelligence, 4507*, 1114-1121.

Lipinski, P. (2008). Evolutionary Decision Support System for Stock Market Trading. *Artificial Intelligence: Methodology, Systems, and Applications, 5253*, 405-409.

Lipinski, P. (2008). Neuro-evolutionary Decision Support System for Financial Time Series Analysis. *International Workshop on Hybrid Artificial Intelligence Systems* (pp. 180-187). Springer, Berlin, Heidelberg.

Lipinski, P. (2010). Frequent Knowledge Patterns in Evolutionary Decision Support Systems for Financial Time Series Analysis. *Natural Computing in Computational Finance, 293*, 131-145.

Lipinski, P. (2010). Frequent Knowledge Patterns in Evolutionary Decision Support Systems for Financial Time Series Analysis. *Natural Computing in Computational Finance, 293*, 131-145.

Lipinski, P., & Korczak, J. J. (2004). Performance measures in an evolutionary stock trading expert system. *International Conference on Computational Science* (pp. 835-842). Springer Berlin Heidelberg.

Lo, A. W., Mamaysky, H., & Wang, J. (2000). Foundations of Technical Analysis: Computational Algorithms, Statistical Inference, and Empirical Implementation. *Journal of Finance, 55*(4), 1705-1765.

Lui, W., & Zheng, W. A. (2011). Stochastic volatility model and technical analysis of stock price. *Acta Mathematica Sinica, 27*(7), 1283-1296.

Luss, R., & D'Aspremont, A. (2015). Predicting abnormal returns from news using text classification. *Quantitative Finance, 15*(6), 999-1012.

Macedo, L. L., Godinho, P., & Alves, M. J. (2017). Mean-semivariance portfolio optimization with multiobjective evolutionary algorithms and technical analysis rules. *Expert Systems with Applications, 79*, 33-43.

Mayer, D., Belward, J., Widell, H., & Burrage, K. (1999). Survival of the fittest - Genetic Algorithms versus evolution strategies in the optimization of systems models. *Agricultural Systems, 60*(2), 113-122.

McLaney, E. J. (2006). *Business Finance: Theory and Practice (7th Edition).* Madrid: Pearson Education.

Melanie, M. (1998). *An Introduction to Genetic Algorithms.* MIT press.

Mittermayer, M. A. (2004). Forecasting Intraday Stock Price Trends with Text Mining Techniques. *System Sciences, 2004. Proceedings of the 37th Annual Hawaii International Conference on* (pp. 1-10). IEEE.

Mohamed, A. R., Dahl, G. E., & Hinton, G. (2012). Acoustic modeling using deep belief networks. *IEEE Transactions on Audio, Speech, and Language Processing, 20*(1), 14-22.

Mohri, M., Rostamizadeh, A., & Talwalkar, A. (2012). *Foundations of Machine Learning.* MIT Press.

Murphy, K. P. (2012). In *Machine Learning, A Probabilistic Perspective* (p. 587). MIT press.

Nan, X., & Sun, X. (2011). Automatic Stock Market Forecasting System Based on Extended Language Template. *Journal of Information and Computational Science, 8*(1), 112-118.

Nicoară, E. S. (2015). Applying Genetic Algorithms to Optimization Problems in Economics. *Economic Insights - Trends and Challenges, 67*, 125-132.

Niu, X. X., & Suen, C. Y. (2012). A novel hybrid CNN–SVM classifier for recognizing handwritten digits. *Pattern Recognition, 45*(4), 1318-1325.

Nóbrega, J. P., & Oliveira, A. L. (2013). Improving the statistical arbitrage strategy in intraday trading by combining extreme learning machine and support vector regression with linear regression models. *Tools with Artificial Intelligence (ICTAI), 2013 IEEE 25th International Conference on* (pp. 182-188). IEEE.

Opelt, A., Pinz, A., Fussenegger, M., & Auer, P. (2006). Generic object recognition with boosting. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 28*(3), 416-431.

OSTC. (2012). Graduate Trading Program.

Ozturk, M., Toroslu, I. H., & Fidan, G. (2016). Heuristic based trading system on Forex data using technical indicator rules. *Applied Soft Computing, 43*, 170-186.

Papadimitriou, T., Gogas, P., & Stathakis, E. (2014). Forecasting energy markets using support vector machines. *Energy Economics, 44*, 135-142.

Pinto, J. M., Neves, R. F., & Horta, N. (2015). Boosting Trading Strategies performance using VIX indicator together with a dual-objective Evolutionary Computation optimizer. *Expert Systems with Applications, 42*(19), 6699-6716.

Platt, J., Cristianini, N., & Shawe-Taylor, J. (2000). Large Margin DAGs for Multiclass Classification. *Advances in Neural Information Processing Systems*, 547-553.

Qiu, M., Song, Y., & Akagi, F. (2016). Application of artificial neural network for the prediction of stock market returns: The case of the Japanese stock market. *Chaos, Solitons and Fractals, 85*, 1-7.

Qu, B., Zhou, Q., Xiao, J., Liang, J., & Suganthan, P. (2017). Large-Scale Portfolio Optimization Using Multiobjective Evolutionary Algorithms and Preselection Methods. *Mathematical Problems in Engineering*, Article ID: 4197914.

Rätsch, G., Onoda, T., & Müller, K. R. (2001). Soft margins for AdaBoost. *Machine Learning, 42*(3), 287-320.

Robert, S. (1990). The strength of weak learnability. *Machine Learning, 5*(2), 197-227.

Salehi, M., Moradi, M., & Molaei, S. (2015). Forecasting systematic risk by Least Angel Regression, AdaBoost and Kernel Ridge Regression. *Modern Applied Science, 9*(11), 135-143.

Schmitt, T. A., Chetalova, D., Schäfer, R., & Guhr, T. (2013). Non-stationarity in financial time series: Generic features and tail behavior. *EPL (Europhysics Letters), 103*(5), 58003.

Sharpe, W. F. (1966). Mutual fund performance. *The Journal of Business, 39*(1), 119-138.

Sharpe, W. F. (1994). The Sharpe Ratio. *The Journal of Portfolio Management, 21*(1), 49-58.

Sharpe, W. F. (1999). In *Investments* (p. 614). Upper Saddle River, NJ: Prentice-Hall.

Smola, A. J., & Schölkopf, B. (2004). A tutorial on support vector regression. *14*(3), 199-222.

Sortino, F. A. (1994). Performance measurement in a downside risk framework. *The Journal of Investing, 3*(3), 59-64.

Stewart, B. (1978). An Analysis of Speculative Trading in Grain Futures. In *USDA Technical Bulletin 1001* (p. 57).

Tang, X. (2009). Hybrid Hidden Markov Model and Artificial Neural Network for Automatic Speech Recognition. *Pacific-Asia Conference on Circuits, Communications and System, 2009. PACCS'09. Pacific-Asia Conference on* (pp. 682-685). IEEE.

Tanveer, M., Shubham, K., Aldhaifallah, M., & Ho, S. S. (2016). An efficient regularized K-nearest neighbor based weighted twin support vector regression. *Knowledge-Based Systems, 94*, 70-87.

Taylor, B. J., Darrah, M. A., & Moats, C. D. (2003). Verification and validation of neural networks: a sampling of research in progress. *AeroSense 2003* (pp. 8-16). International Society for Optics and Photonics.

Ticknor, J. L. (2013). A Bayesian regularized artificial neural network for stock market forecasting. *Expert Systems with Applications, 40*(14), 5501-5506.

Ting, K. M. (2010). Accuracy. In *Encyclopedia of Machine Learning* (pp. 9-10). Springer Science & Business Media.

Ting, K. M. (2010). F1-Measure. In *Encyclopedia of Machine Learning* (p. 397). Springer Science & Business Media.

Ting, K. M. (2010). Negative Predictive Value. In *Encyclopedia of Machine Learning* (pp. 715-716). Springer Science & Business Media.

Ting, K. M. (2010). Precision and Recall. In *Encyclopedia of Machine Learning* (p. 781). Springer Science & Business Media.

Ting, K. M. (2010). Specificity. In *Encyclopedia of Machine Learning* (p. 907). Springer Science & Business Media.

Treanor, J. (2015, April 22). *The 2010 'flash crash': how it unfolded*. Retrieved 08 13, 2017, from The Guardian: https://www.theguardian.com/business/2015/apr/22/2010-flash-crash-new-york-stock-exchange-unfolded

Triennial Central Bank Survey of foreign exchange and OTC derivatives markets in 2016. (2016). Retrieved January 22, 2017, from Bank for International Settlements: http://www.bis.org/publ/rpfx16.htm

Tsai, P. Y., Hsu, Y., Chiu, C. T., & Chu, T. T. (2015). Accelerating AdaBoost algorithm using GPU for multi-object recognition. *Circuits and Systems (ISCAS), 2015 IEEE International Symposium on* (pp. 738-741). Lisbon: IEEE.

Vapnik, V. (2013). *The Nature of Statistical Learning Theory.* New York: Springer science & business media.

Vezhnevets, A., & Barinova, O. (2007). Avoiding Boosting Overfitting by Removing Confusing Samples. *Machine Learning: ECML 2007*, (pp. 430-441). Warsaw.

Viola, P., & Jones, M. (2001). Rapid object detection using a boosted cascade of simple features. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on.* IEEE.

Wang, L., & Wu, J. (2011). Neural Network Ensemble Model Using PPR and LS-SVR for Stock Market Forecasting. *nternational Conference on Intelligent Computing* (pp. 1-8). Springer, Berlin, Heidelberg.

Wang, L., An, H., Liu, X., & Huang, X. (2016). Selecting dynamic moving average trading rules in the crude oil futures market using a genetic approach. *Applied Energy, 162*, 1608-1618.

Wang, L., An, H., Xia, X., Liu, X., Sun, X., & Huang, X. (2014). Generating moving average trading rules on the oil futures market with genetic algorithms. *Mathematical Problems in Engineering*, Article ID: 101808.

Watson, P. K., & Teelucksingh, S. S. (2002). *A Practical Introduction to Econometric Methods: Classical and Modern.* University of the West Indies Press.

Wearden, G., Teather, D., & Treanor, J. (2008, September 15). *Banking crisis: Lehman Brothers files for bankruptcy protection*. Retrieved December 31, 2017, from The Guardian.

Wen, Q., Yang, Z., Song, Y., & Jia, P. (2010). Automatic stock decision support system based on box theory and SVM algorithm. *Expert Systems with Applications, 37*(2), 1015-1022.

Whitley, D., Hains, D., & Howe, A. (2010). A Hybrid Genetic Algorithm for the Traveling Salesman Problem Using Generalized Partition Crossover. *Parallel Problem Solving from Nature, PPSN XI*, (pp. 566-575).

Winston, P. (2010). *6.034 Artificial Intelligence, Fall 2010.* Retrieved June 5, 2014, from MIT OpenCourseWare: Massachusetts Institute of Technology (License: Creative Commons BY-NC-SA): http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-034-artificial-intelligence-fall-2010

Ye, F., Zhang, L., Zhang, D., Fujita, H., & Gong, Z. (2016). A novel forecasting method based on multi-order fuzzy time series and technical analysis. *Information Sciences, 367*, 41-57.

Yu, L., Wang, S., & Lai, K. (2005). Mining stock market tendency using GA-based support vector machines. *Internet and Network Economics*, (pp. 336-345).

Yümlü, S., Gürgen, F. S., & Okay, N. (2005). A comparison of global, recurrent and smoothed-piecewise neural models for Istanbul stock exchange (ISE) prediction. *Pattern Recognition Letters, 26*(13), 2093-2103.

Zapranis, A., & Tsinaslanidis, P. (2010). Identification of the Head-and-Shoulders Technical Analysis Pattern with Neural Networks. *International Conference on Artificial Neural Networks* (pp. 130-136). Springer, Berlin, Heidelberg.

Zhai, Y., Hsu, A., & Halgamuge, S. K. (2007). Combining News and Technical Indicators in Daily Stock Price Trends Prediction. *International Symposium on Neural Networks* (pp. 1087-1096). Springer, Berlin, Heidelberg.

Zhang, G. (2003). Time series forecasting using a hybrid ARIMA and neural network model. *Neurocomputing, 50*, 159-175.

Zhang, X., Hu, Y., Xie, K., Wang, S., Ngai, E., & Liu, M. (2014). A causal feature selection algorithm for stock prediction modeling. *Neurocomputing, 142*, 48-59.

Zhang, Y., & Wu, L. (2009). Stock market prediction of S&P 500 via combination of improved BCO approach and BP neural network. *Expert Systems with Applications, 36*(5), 8849-8854.

Zhao, Y., Xue, J., & Chen, X. (2015). Ensemble Learning Approaches in Speech Recognition. In *Speech and Audio Processing for Coding, Enhancement and Recognition* (pp. 113-152). Springer New York.

Zhou, C., Wang, L., Zhang, Q., & Wei, X. (2013). Face recognition based on PCA image reconstruction and LDA. *Optik-International Journal for Light and Electron Optics, 124*(22), 5599-5603.

Appendix A – Further Genetic Algorithm results

This appendix presents further results relating to Table 15 in Section 6.3.3, where each series represents a different Genetic Algorithm configuration. Figures 190, 191 and 192 show the average maximum fitness value of 100 Genetic Algorithm runs using a population of trading strategies consisting of 6 technical analysis interpretations

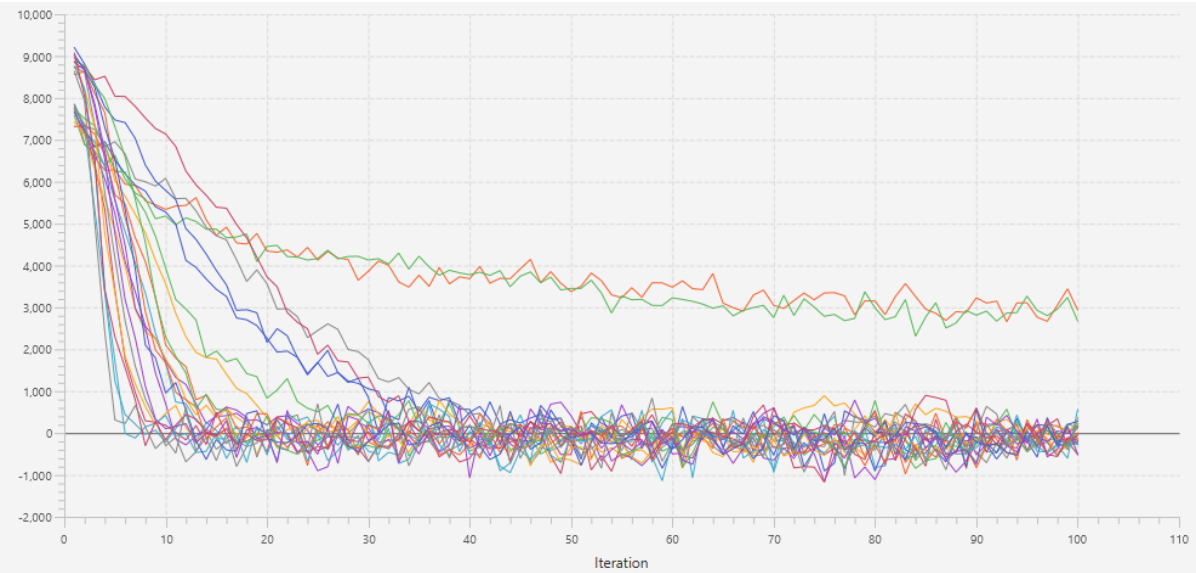


Figure 190: Genetic Algorithm results obtained from insample data

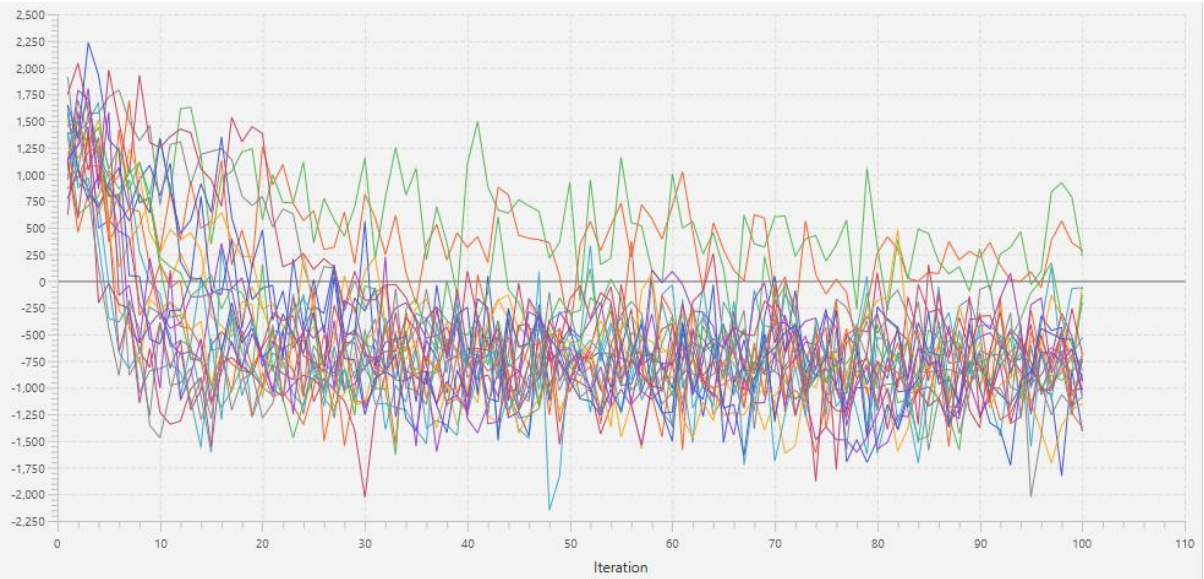


Figure 191: Genetic Algorithm results obtained from validation data

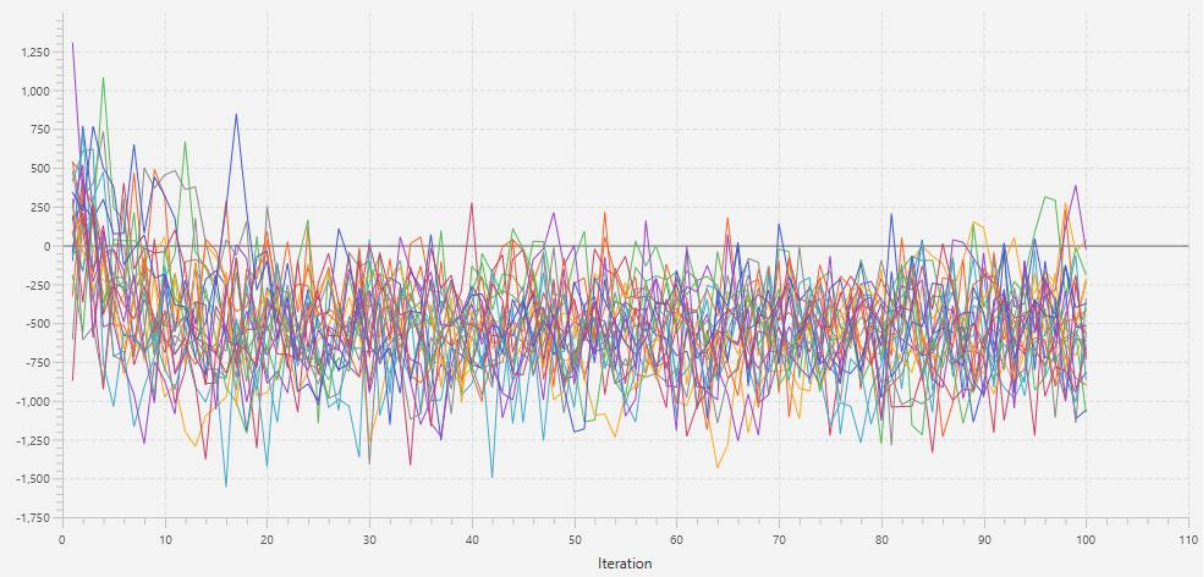


Figure 192: Genetic Algorithm results obtained from outsample data

Figures 193, 194 and 195 show the average maximum fitness value of 100 Genetic Algorithm runs using a population of trading strategies consisting of 7 technical analysis interpretations.

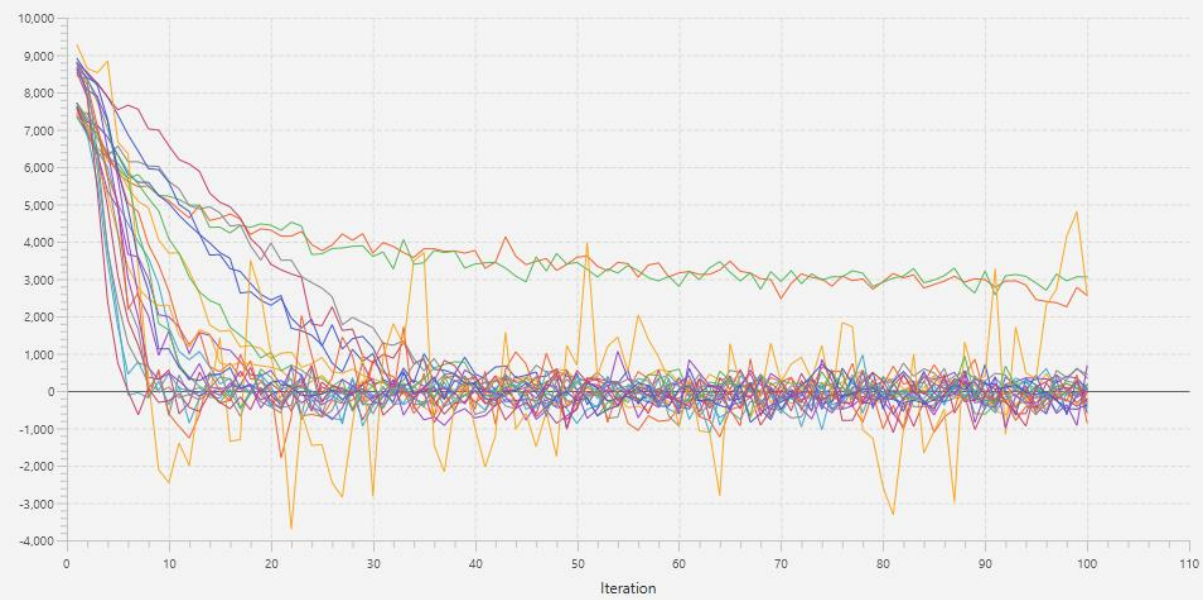


Figure 193: Genetic Algorithm results obtained from insample data

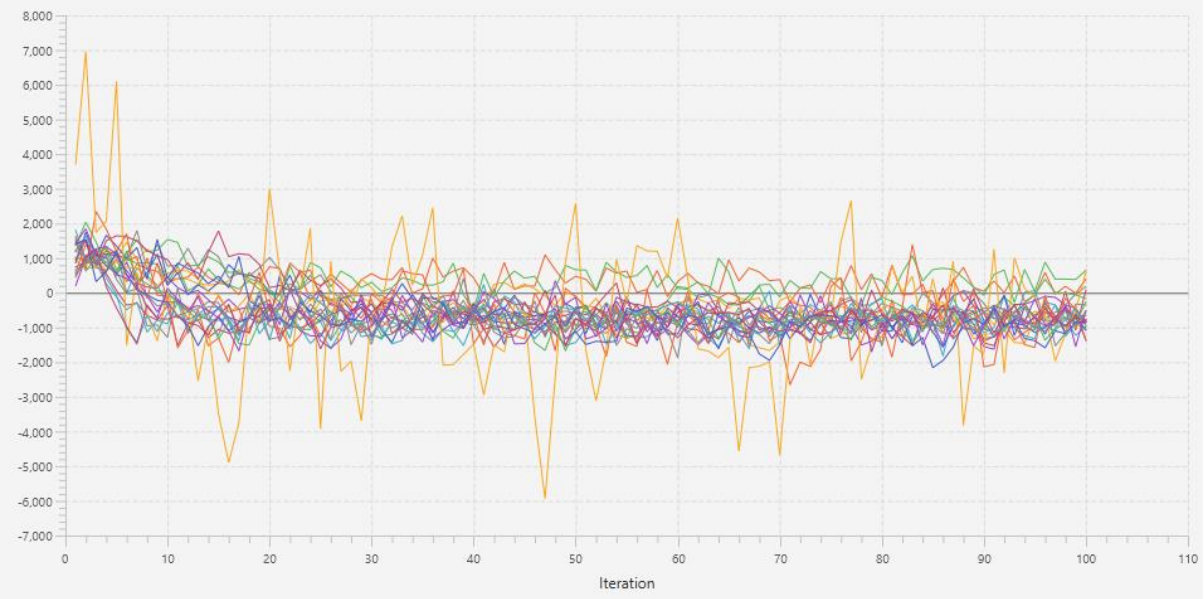


Figure 194: Genetic Algorithm results obtained from validation data

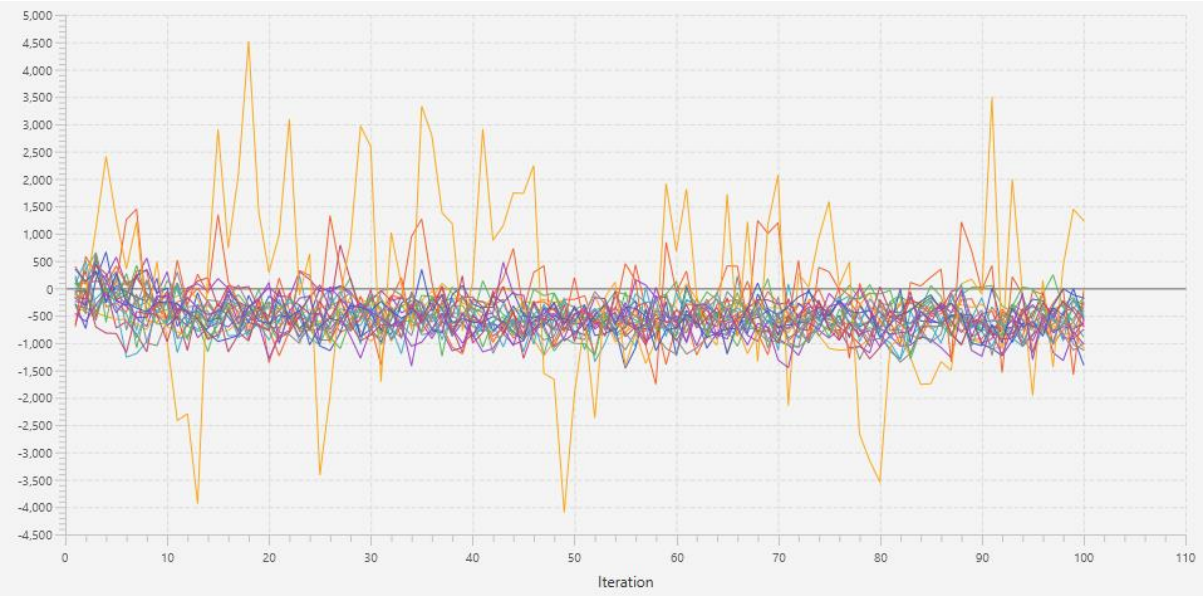


Figure 195: Genetic Algorithm results obtained from outsample data

Appendix B – Classification System Performance Results

This appendix presents the full performance metric results for classification systems produced in Section 11.5, in relation to different trading strategy categorisations.

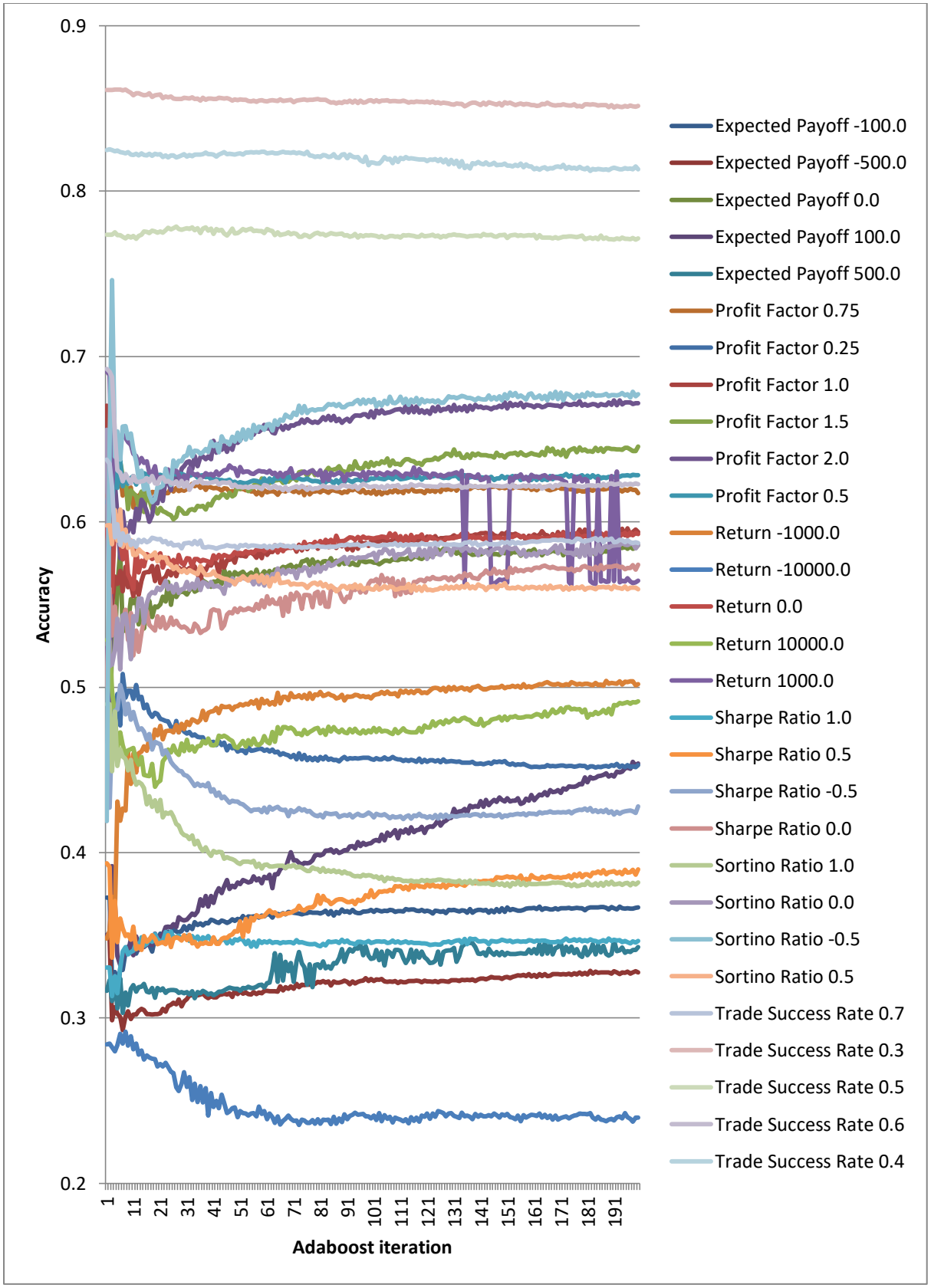


Figure 196: Accuracy of classification systems on validation data using different categorisation metrics and values

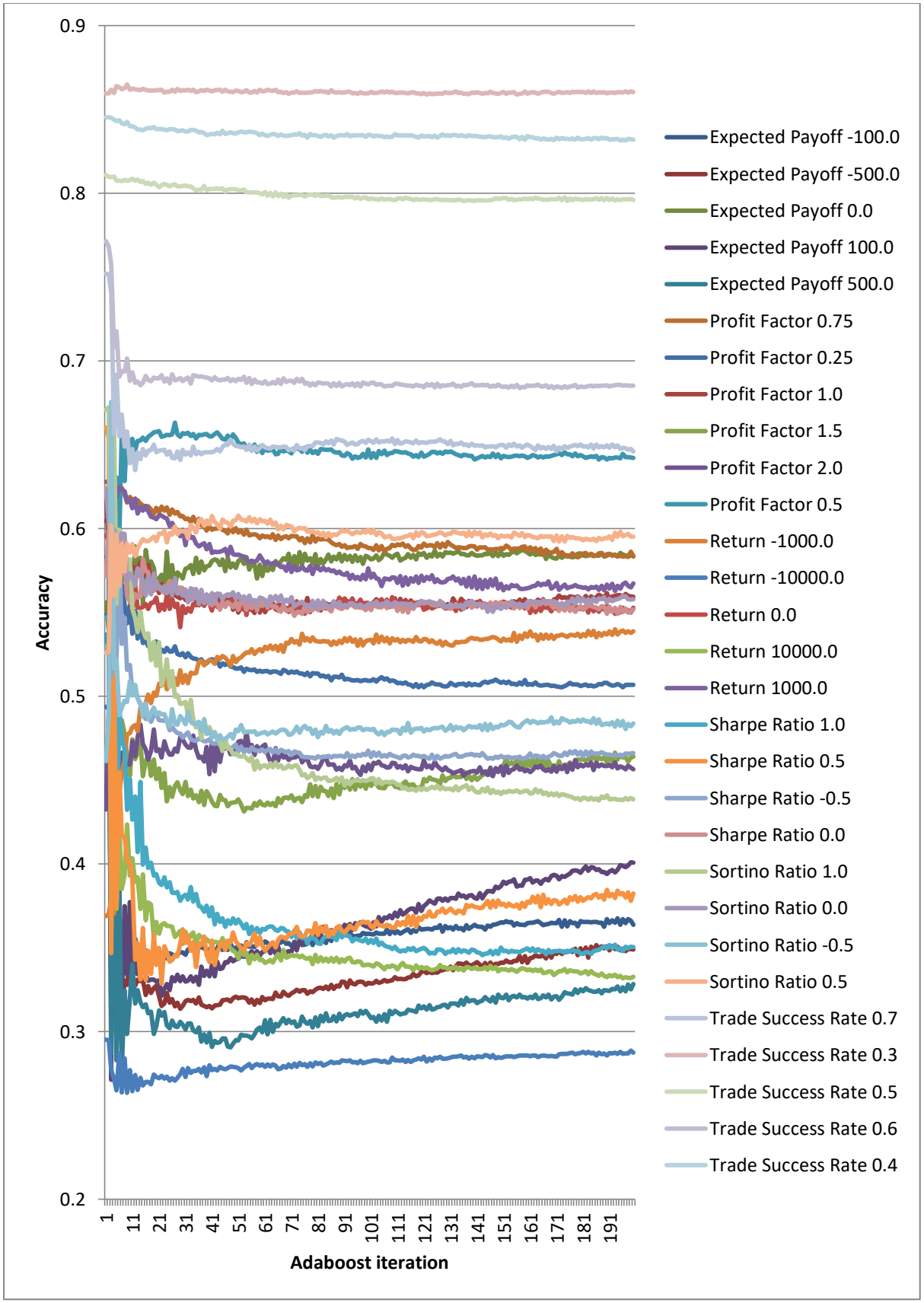


Figure 197: Accuracy of classification systems on outsample data using different categorisation metrics and values

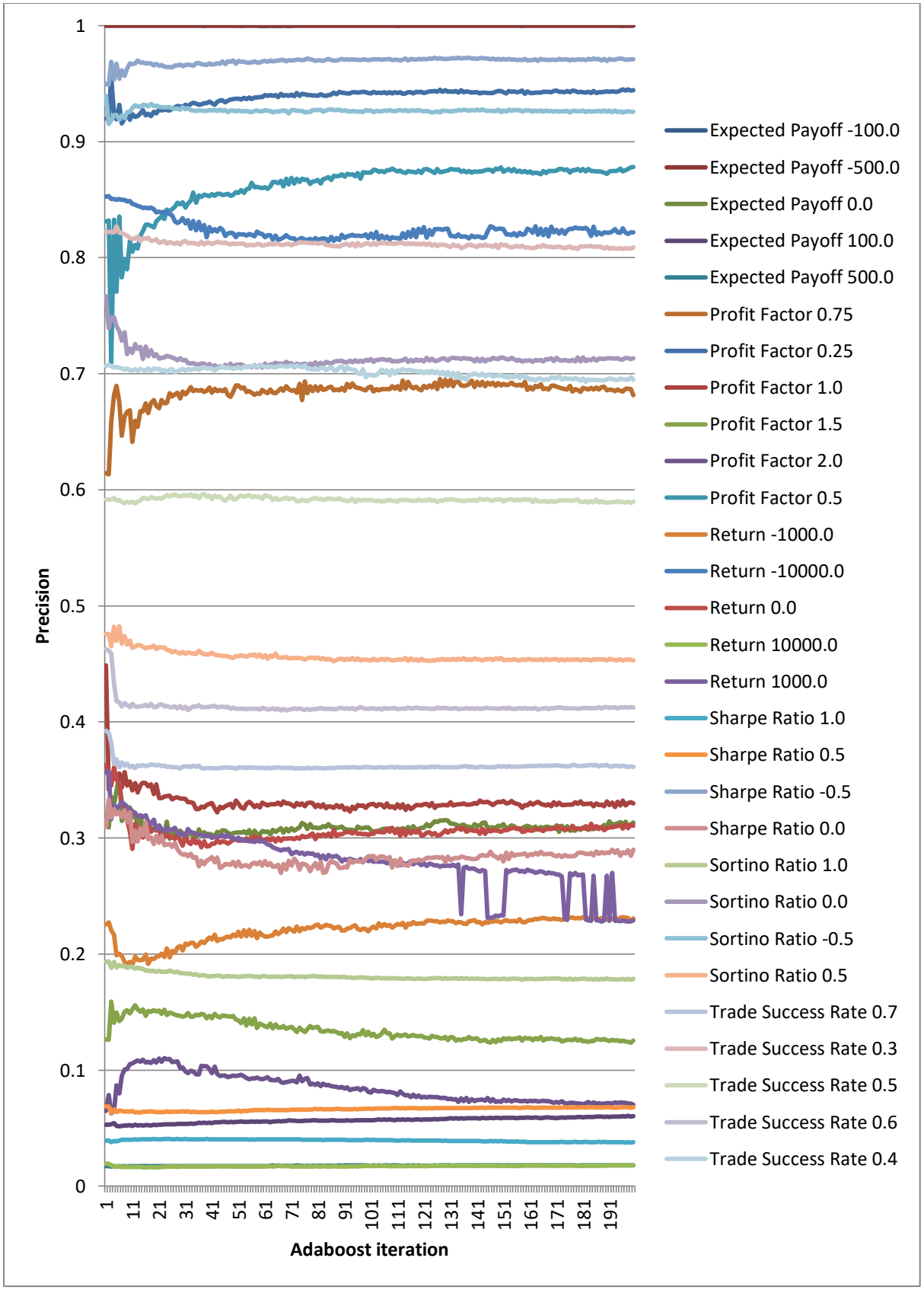


Figure 198: Precision of classification systems on validation data using different categorisation metrics and values

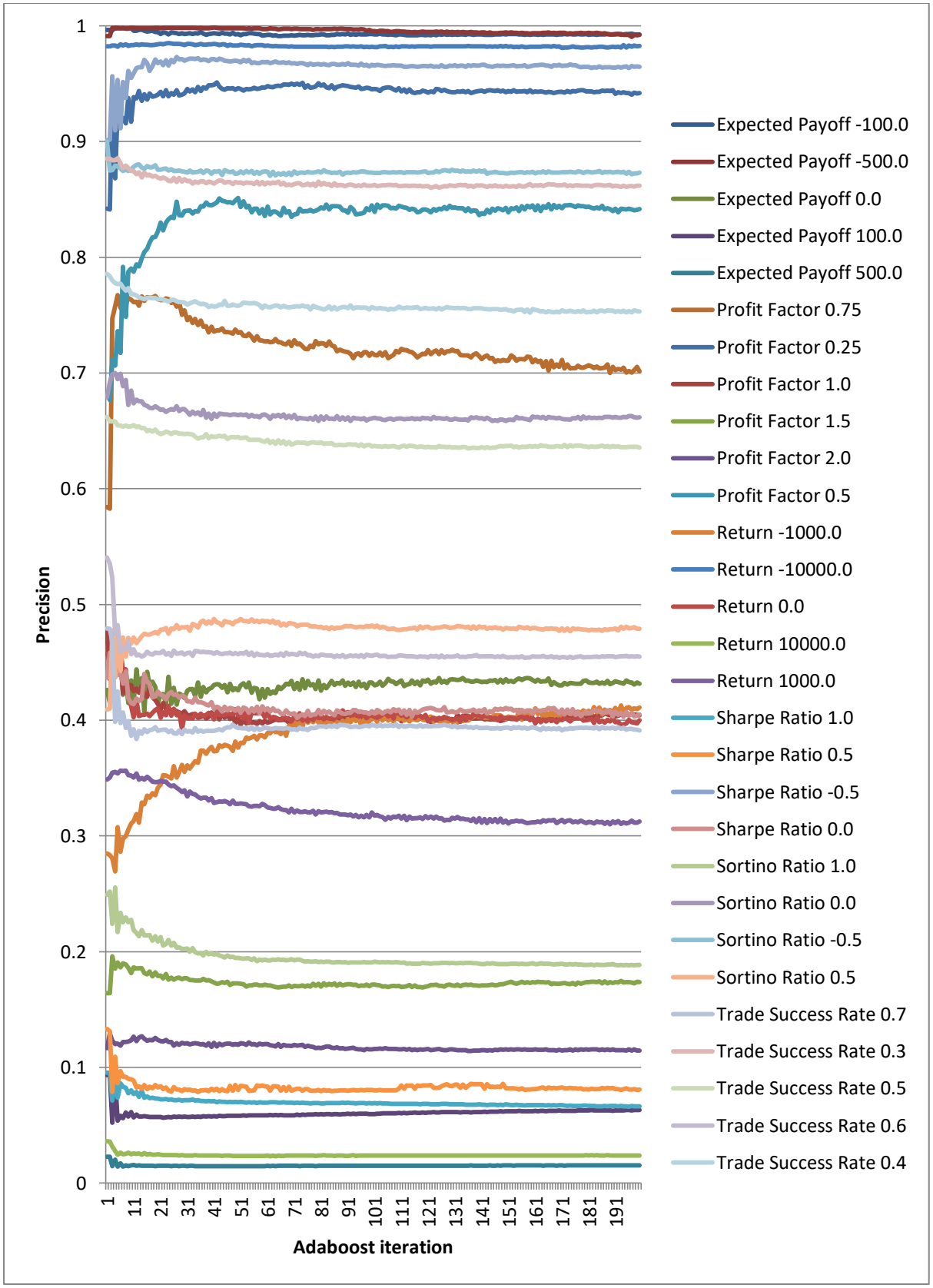


Figure 199: Precision of classification systems on outsample data using different categorisation metrics and values

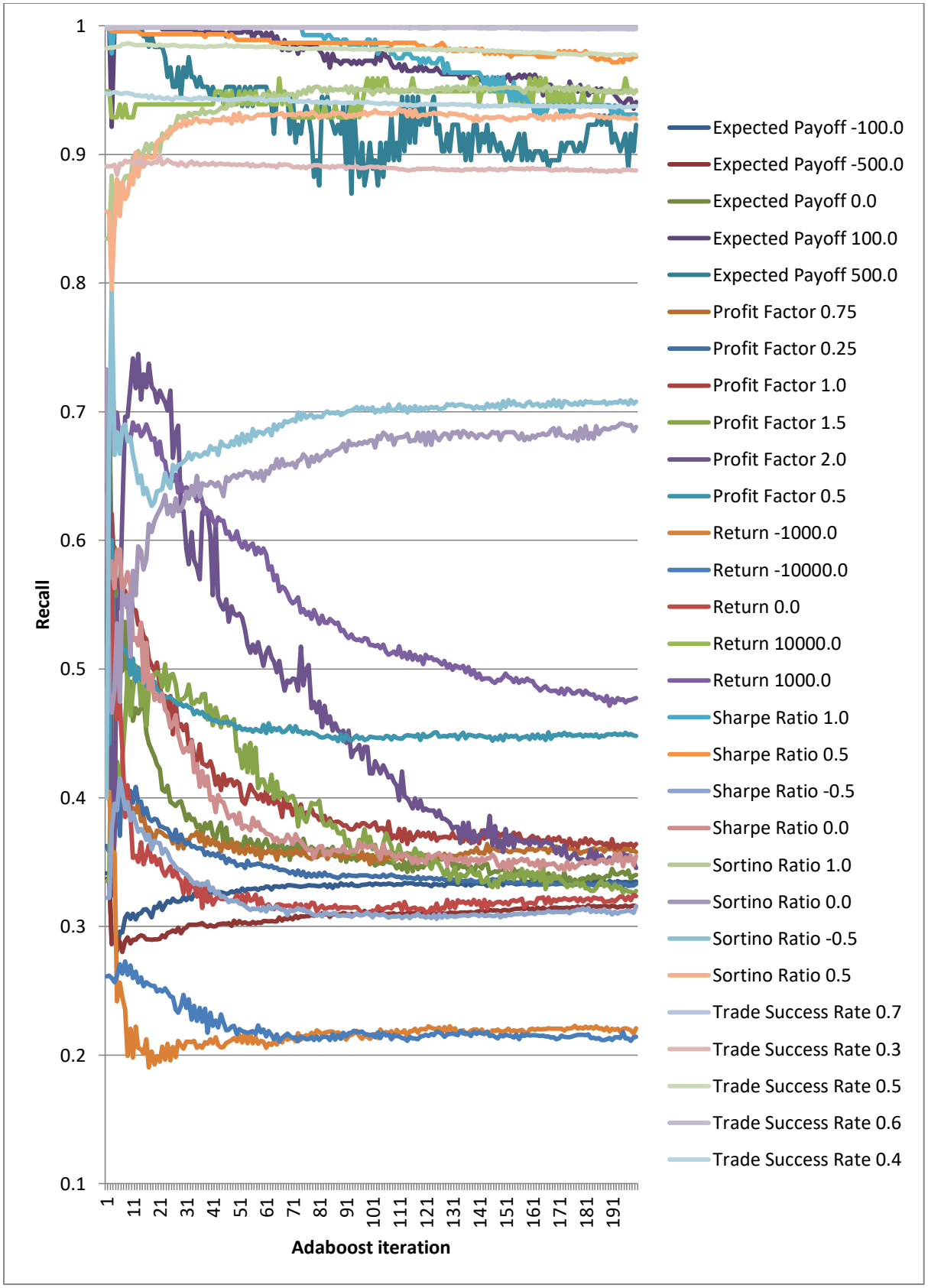


Figure 200: Recall of classification systems on validation data using different categorisation metrics and values

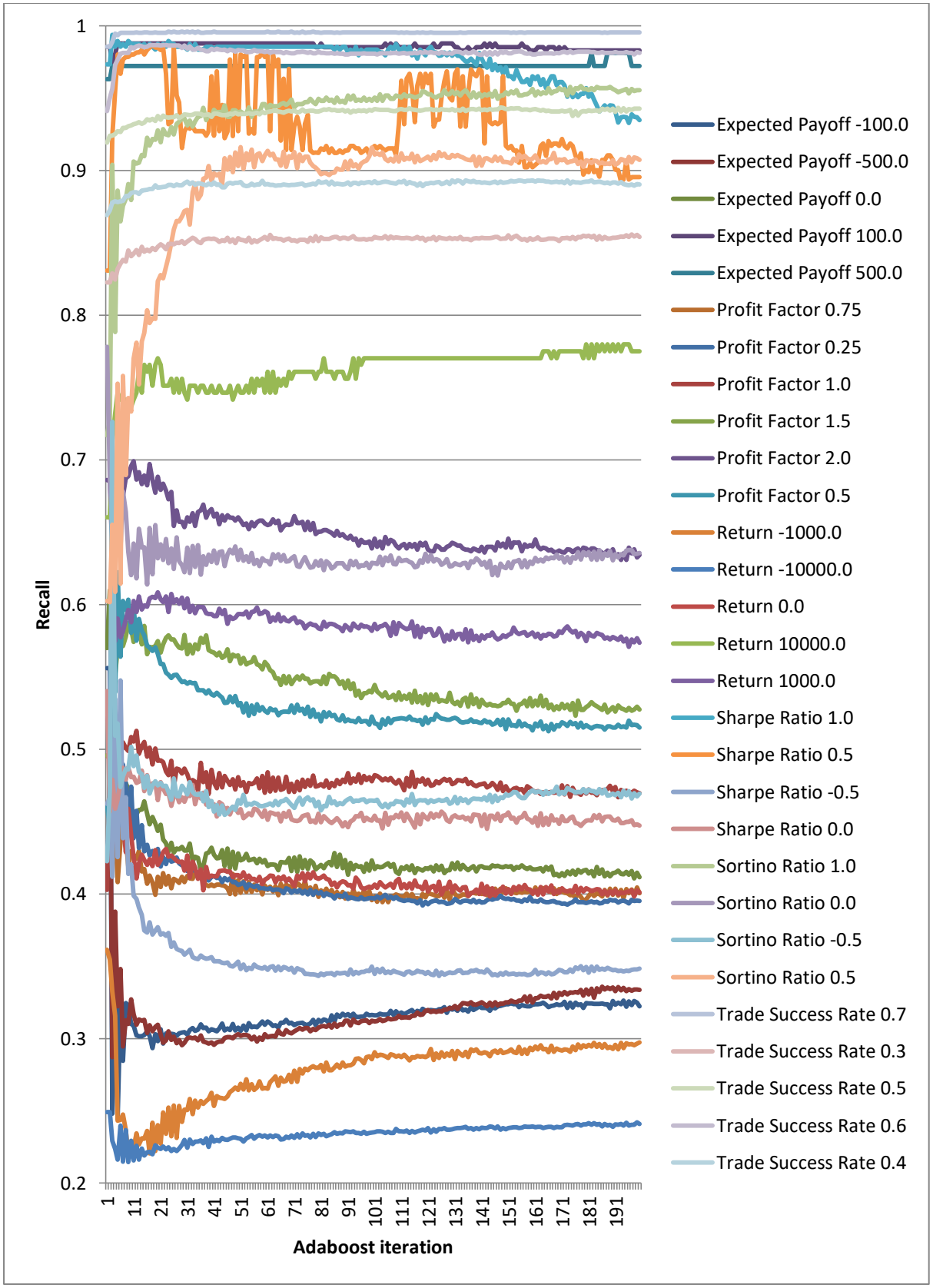


Figure 201: Recall of classification systems on outsample data using different categorisation metrics and values

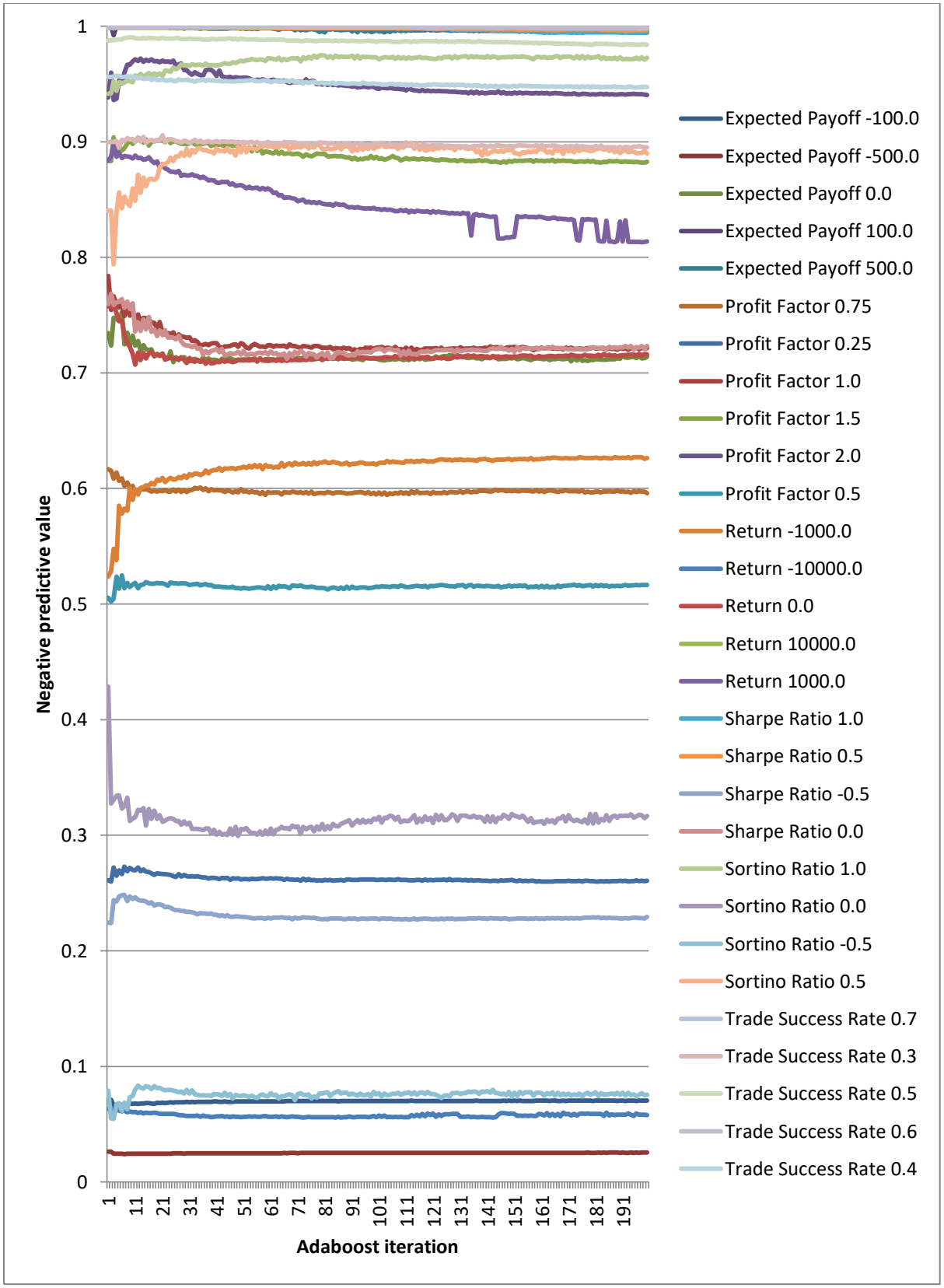


Figure 202: Negative predictive value of classification systems on validation data using different categorisation metrics and values

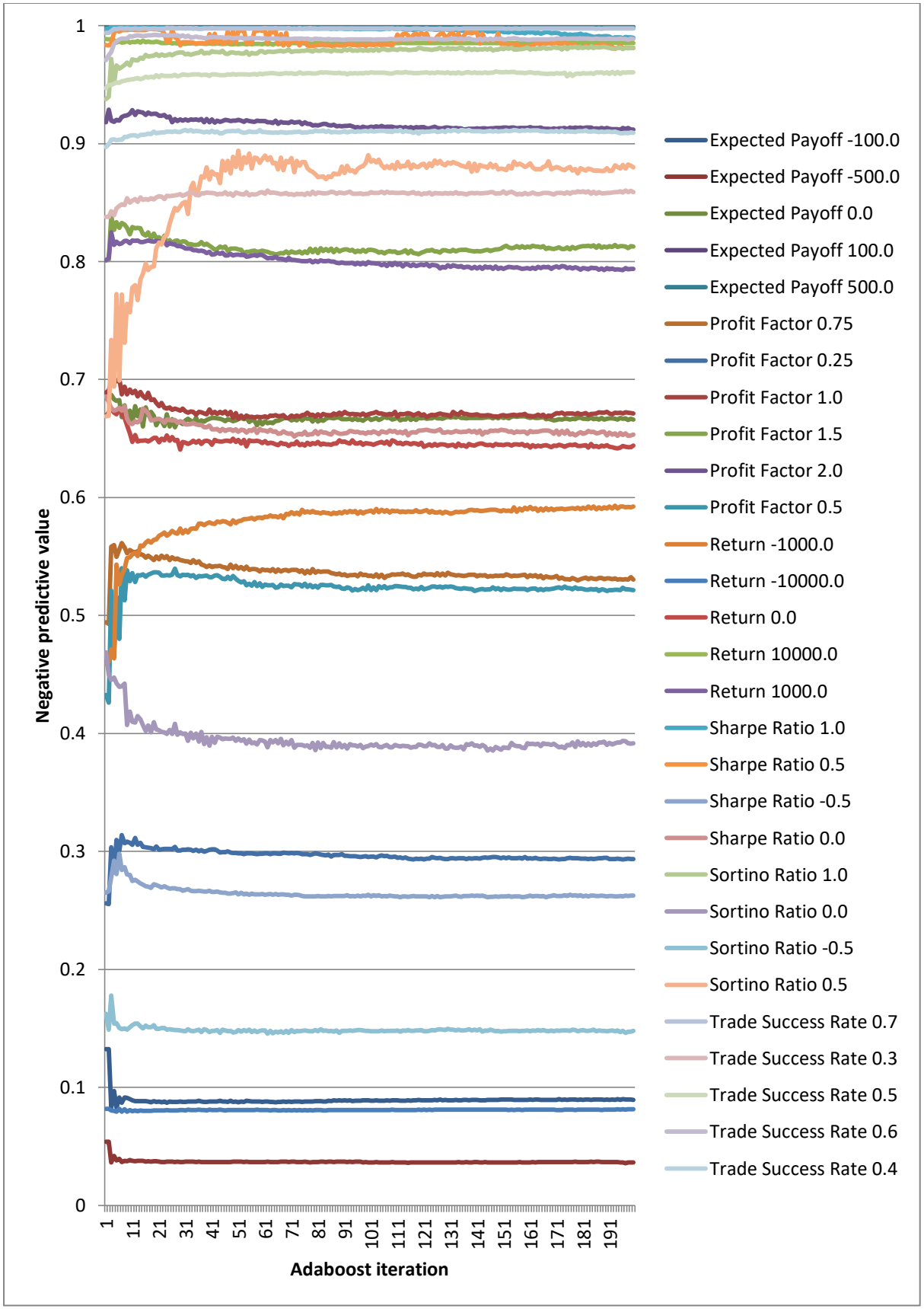


Figure 203: Negative predictive value of classification systems on outsample data using different categorisation metrics and values

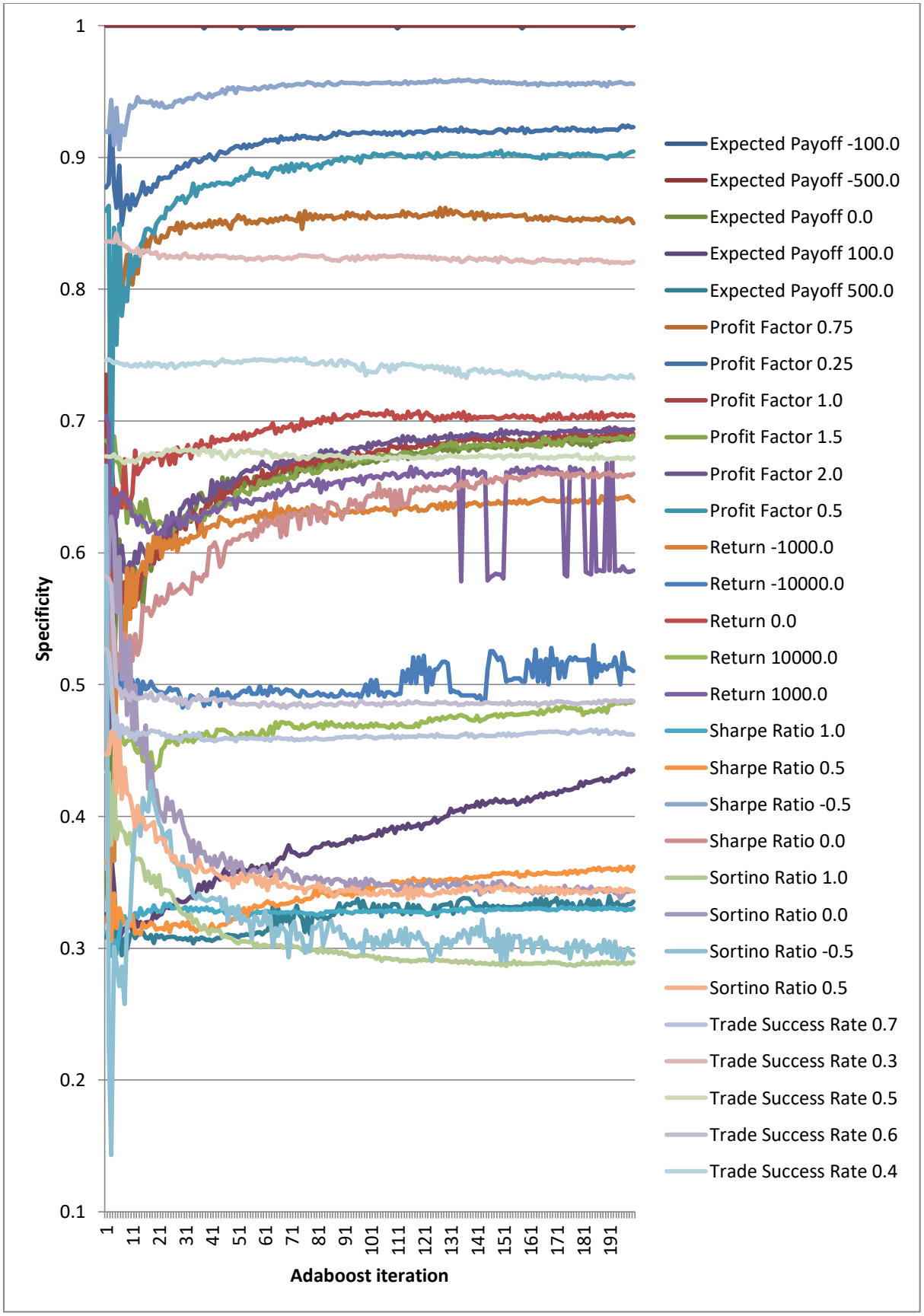


Figure 204: Specificity of classification systems on validation data using different categorisation metrics and values

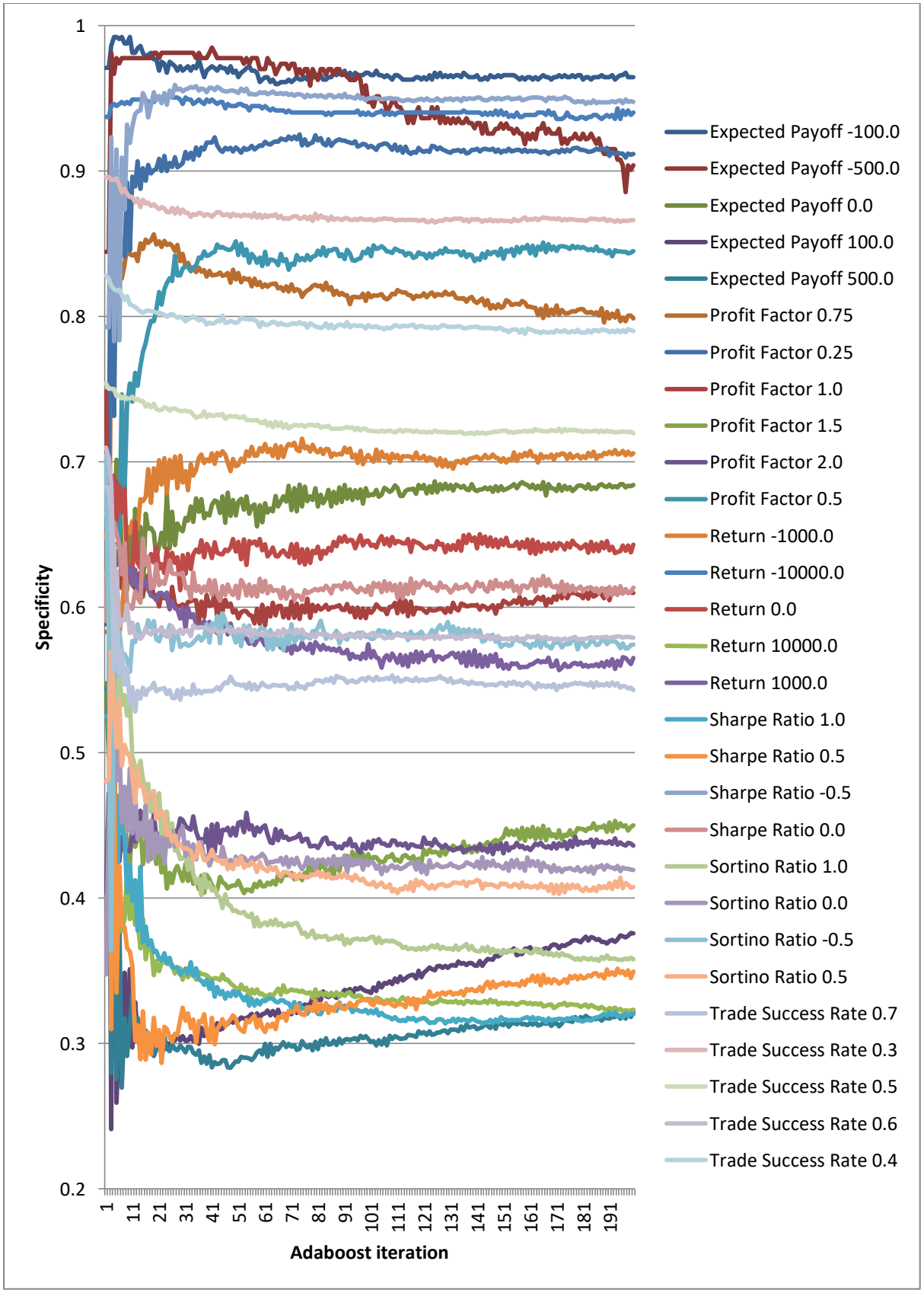


Figure 205: Specificity of classification systems on outsample data using different categorisation metrics and values

Appendix C – MaTool

MaTool is a program that was developed by the current author, which explores and encapsulates most of the research undertaken during this PhD into a graphical user interface. MaTool is split into many workspaces and the main features are outlined in the following sections.

Market Workspace

This part of the software allows the user to access and examine market data such as share and option prices. The market data used in this thesis was downloaded from Forex's MetaTrader 4 application.

Main functionality:

1. Visualise and load market data (see Figure 206). *Market data can be loaded into MaTool via the 'File' menu*
2. Ability to see technical analysis information (see Figure 207). *Technical analysis algorithms can be shown via the 'Tools' menu*
3. Ability to zoom in and out and adjust the x-axis by using the mouse
4. Ability to overlay trading strategy decisions on the market data (see Figure 208). *The trading strategy can be loaded in via the 'Market' menu*
5. Ability to view the perfect buy and sell decisions for short term and long term trends (see Figure 209). *This can be shown via the 'Market' menu and selecting the 'Show trends' menu item*

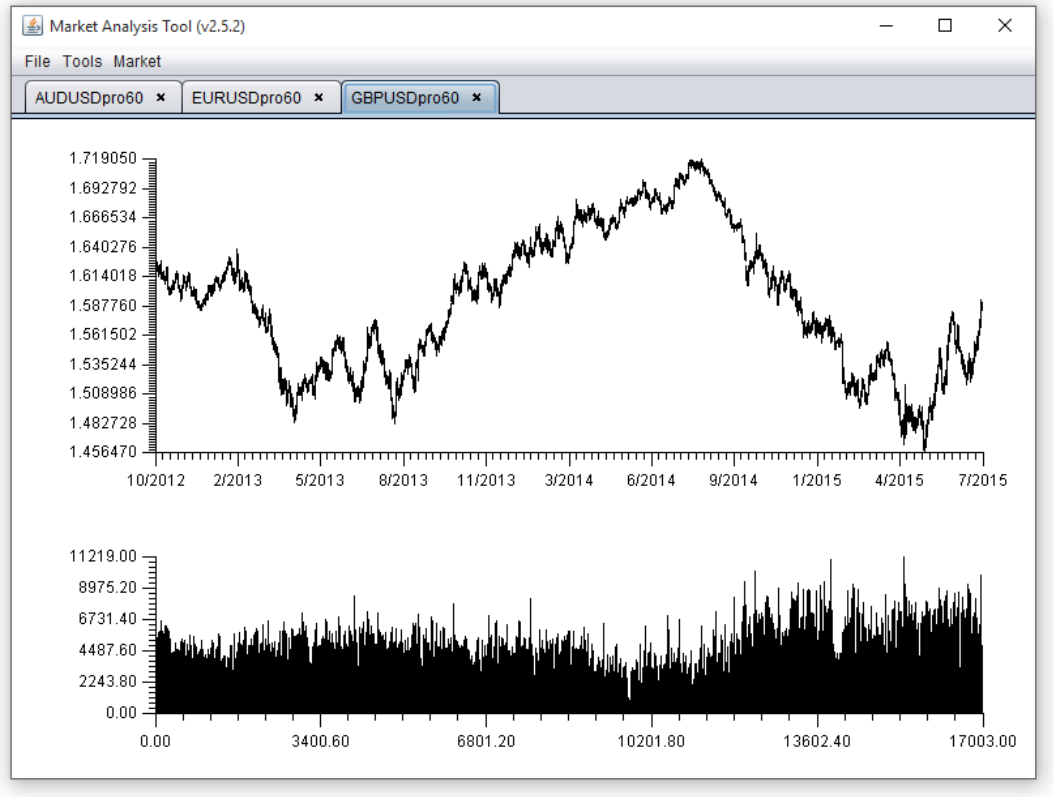


Figure 206: Market workspace loaded with FOREX exchange rate market data

Figure 207: Technical analysis information on AUDUSD market data

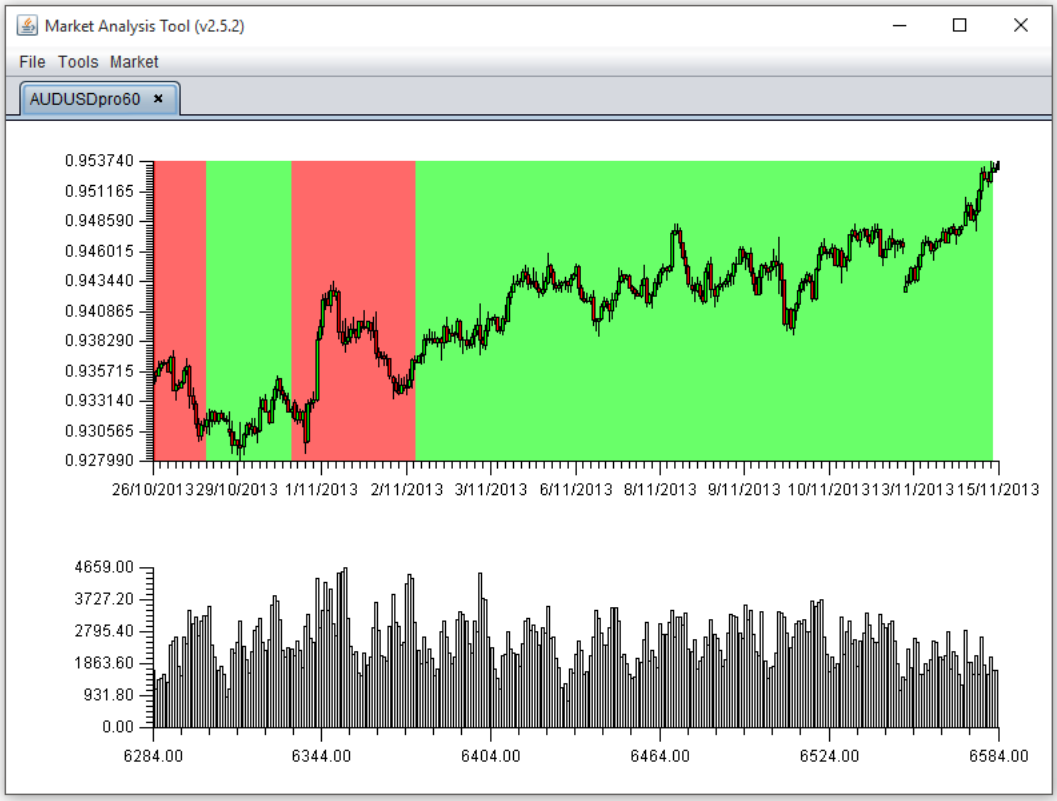


Figure 208: A loaded trading strategy's buy (green) and sell (red) decisions

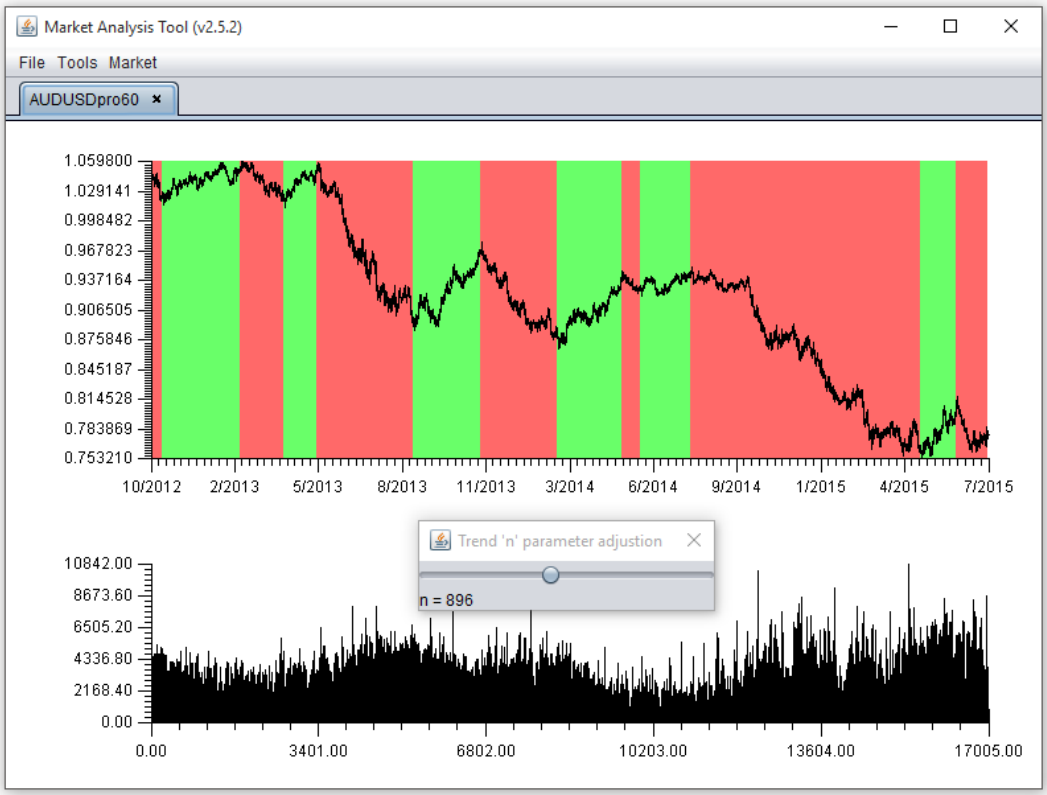


Figure 209: 'Perfect' buy and sell decisions with a dialog which adjusts the view from short term to long term decisions

Model Workspace

This part of the software allows the user to manipulate and inspect trading strategies (see Figure 210).

Main functionality:

1. Load and save trading strategies. *This is achieved via the 'File' menu*
2. Manually create or edit trading strategies using the mouse
3. Inspect trading strategies and their components. *This is done by double clicking a node of the trading strategy; if it was the root node then the user would be inspecting the whole trading strategy*
4. View the equity curve of a trading strategy and its components (see Figure 211). *This can be done by first selecting a node or nodes then selecting 'Show equity curve' via the 'Tools' menu*

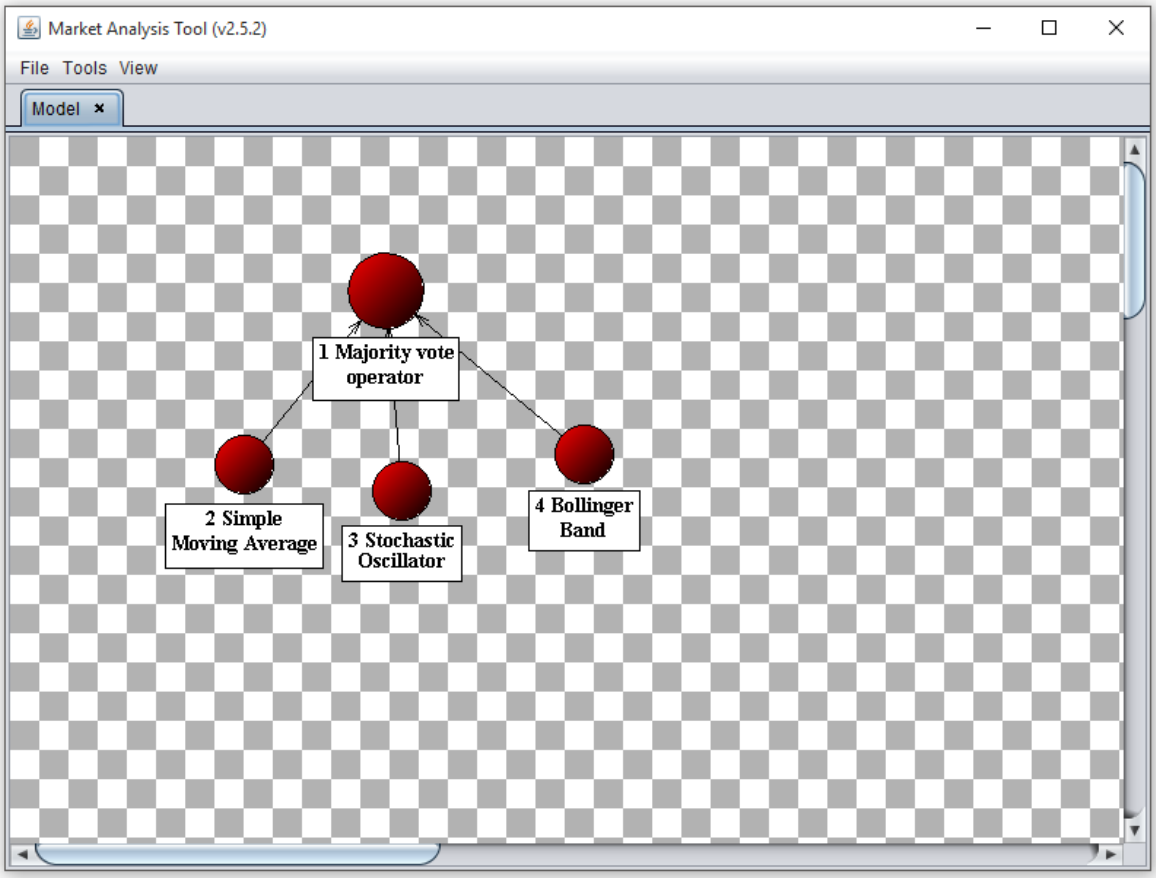


Figure 210: Model workspace showing a trading strategy

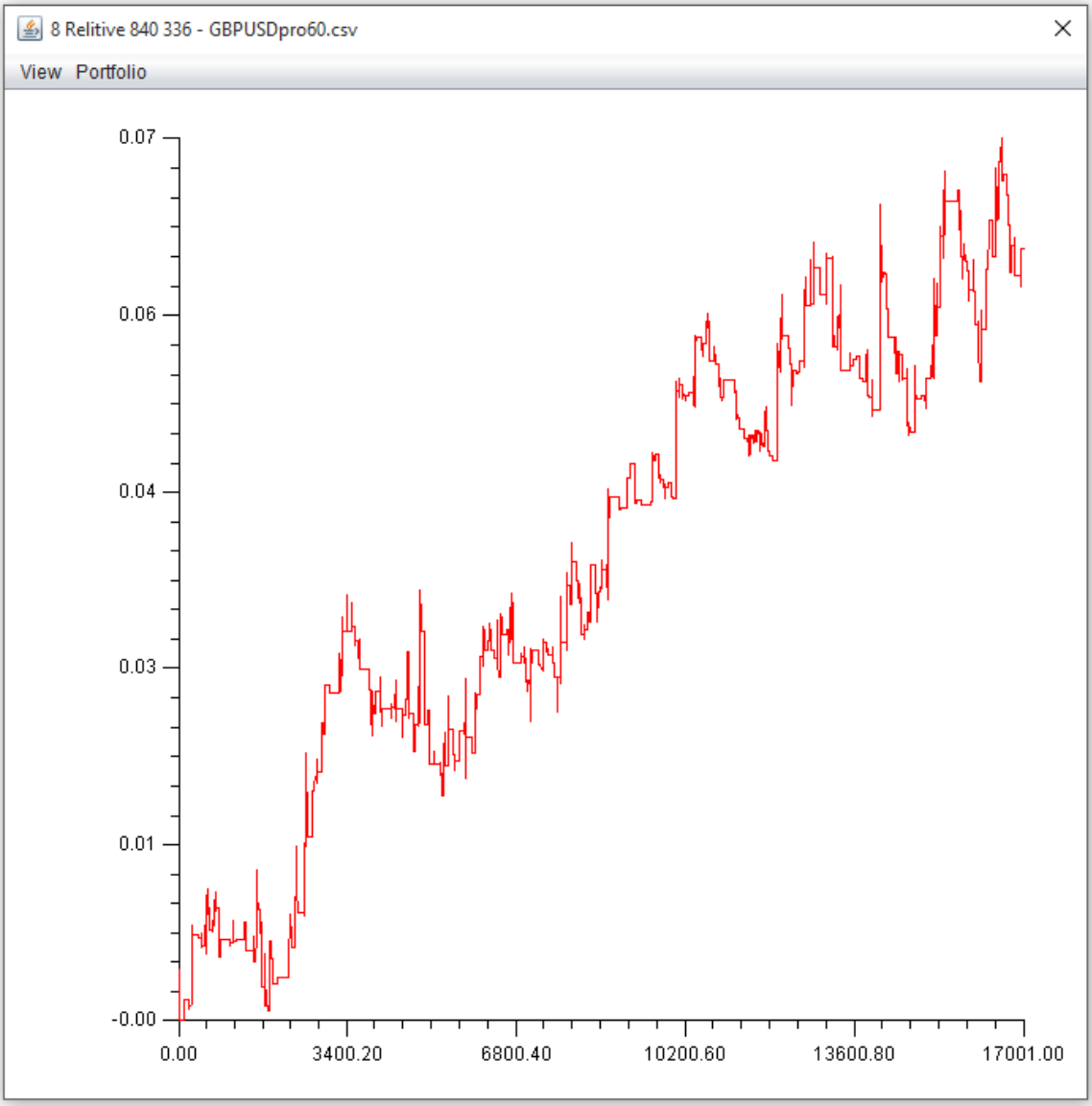


Figure 211: Equity curve of a trading strategy algorithm

Optimisation Workspace

This part of the software finds the 'best' parameter settings for technical analysis algorithms given a user specified objective function (see Figure 212).

Main functionality:

1. Allows the user to find the best technical analysis parameter settings
2. Allows the user to create a weighted objective function so that it can maximise it when finding the best parameter settings
3. Allows the user to specify constraints which must be satisfied
4. Allows the user to save the technical analysis algorithms and save the objective function with constraints. *This can be done via the 'File' menu*

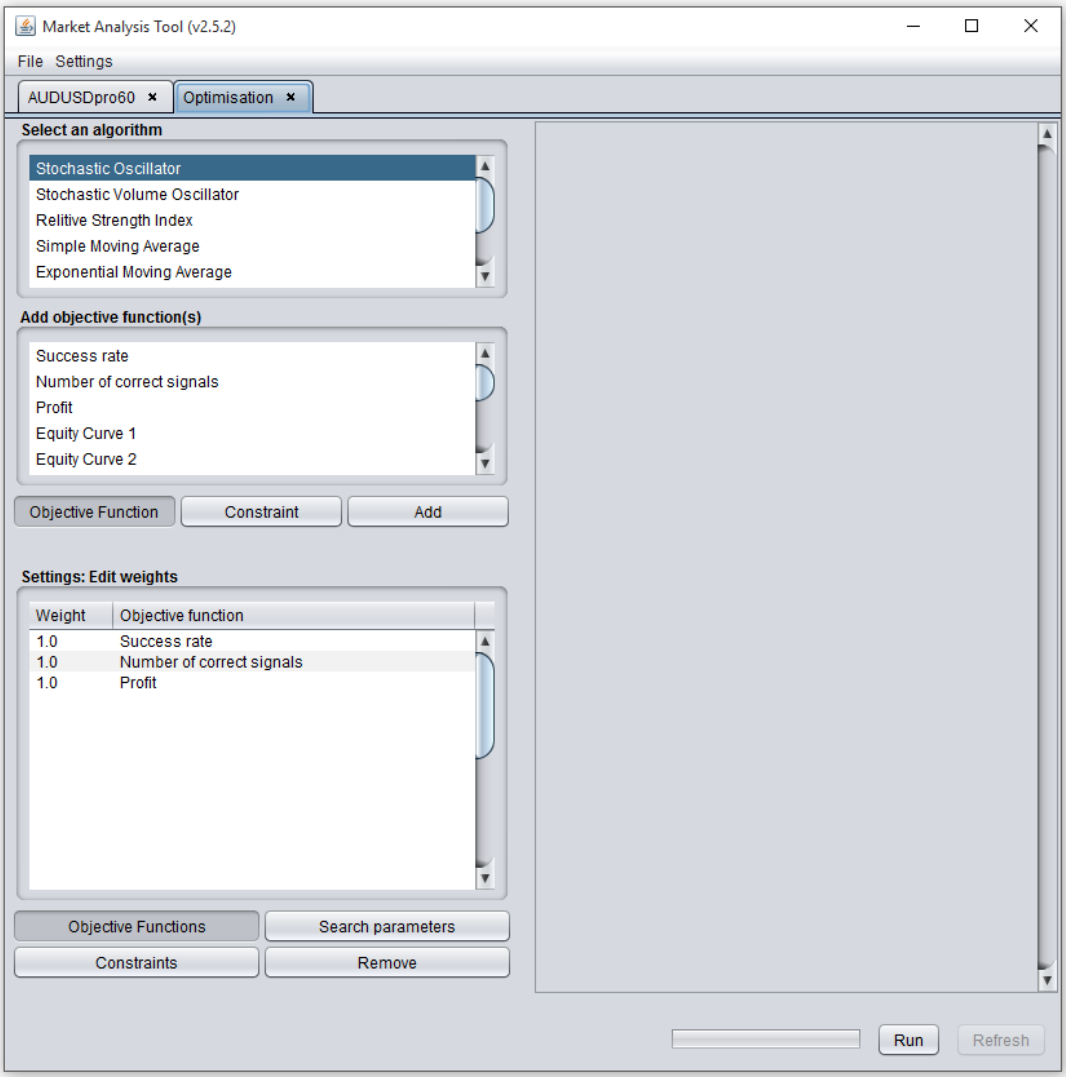


Figure 212: Technical analysis parameter optimisation

Trading Strategy Manager

This part of the software visualises a collection of trading strategies using three dimensional scatter graphs with user specified axis variables (see Figure 213). The main purpose was to give insights into the relationship between different metrics and currently only works with older directories.

Main functionality:

- Creates and saves trading strategies
- Creates and saves boosted classifiers
- Allows the user to create on three dimensional graph scatter plots of trading strategies which depend on user specified metrics

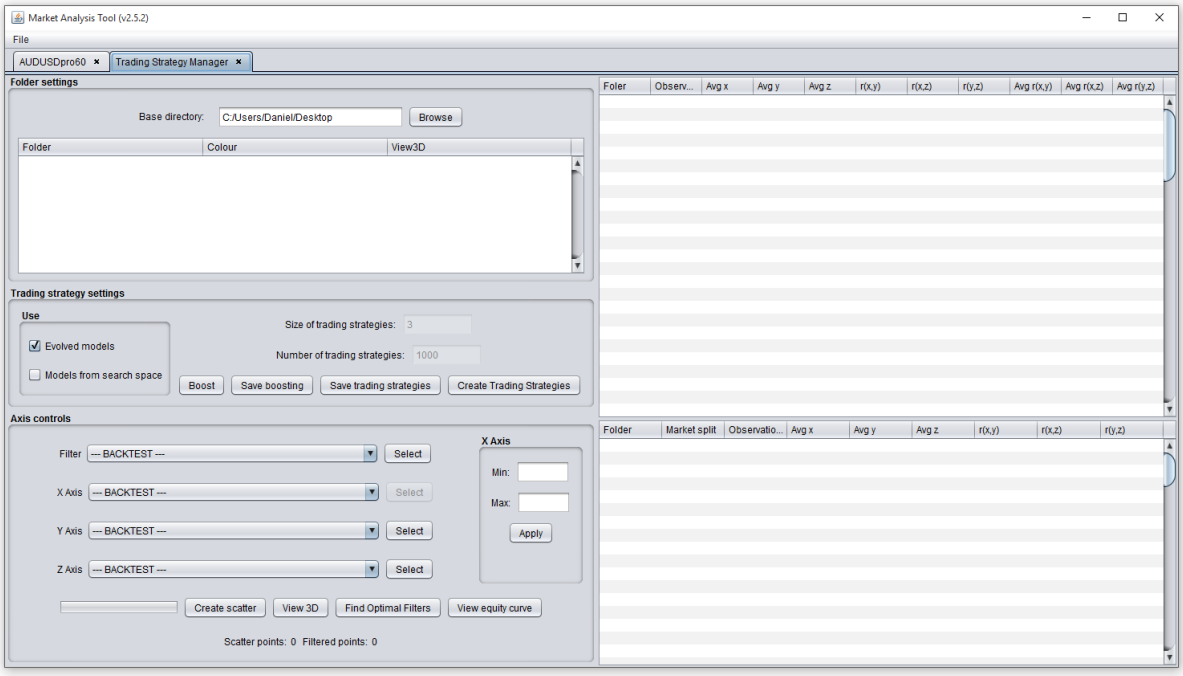


Figure 213: Trading strategy manager workspace

Monte Carlo Simulation Workspace

This part of the software gives the user access to artificial market data which can be used in Monte Carlo simulation tests (see Figure 214).

Main functionality:

- Allows the user to specify the number of simulation days
- Allows the user to specify the number of simulations
- Allows the user to save the artificial data
- Simulates market data using the conditional discrete price change distribution
- Visually shows the artificial market data simulation

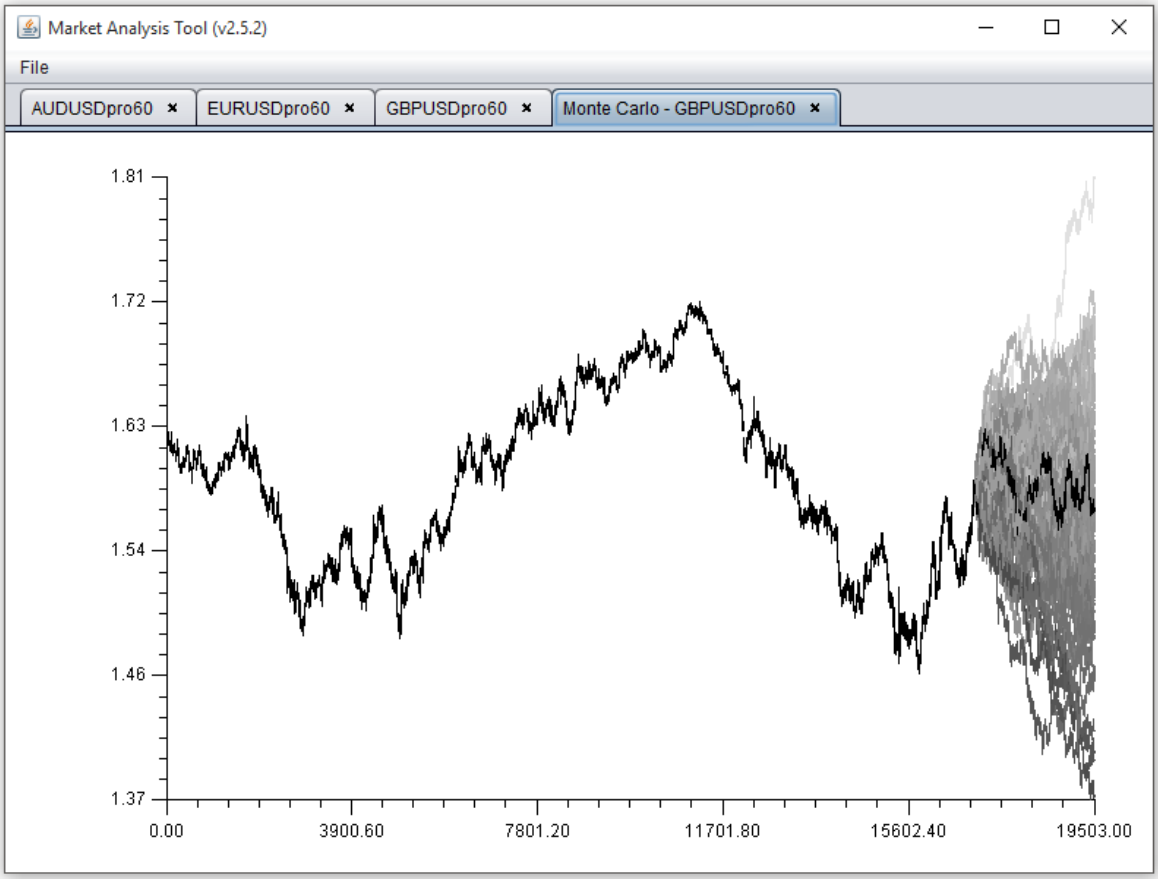


Figure 214: Simulations of market data

Adaboost Workspace

This part of the software contains functionality relating to the investigation and application of Adaboost to classifying trading strategies (see Figures 215 and 216).

Main functionality:

- Plots the classification performance of created and constructing strong classifiers on insample, validation and outsample data
- Plots the classification performance of the bootstrap classifier made up of many strong classifiers
- Reports the results of the bootstrap classifier

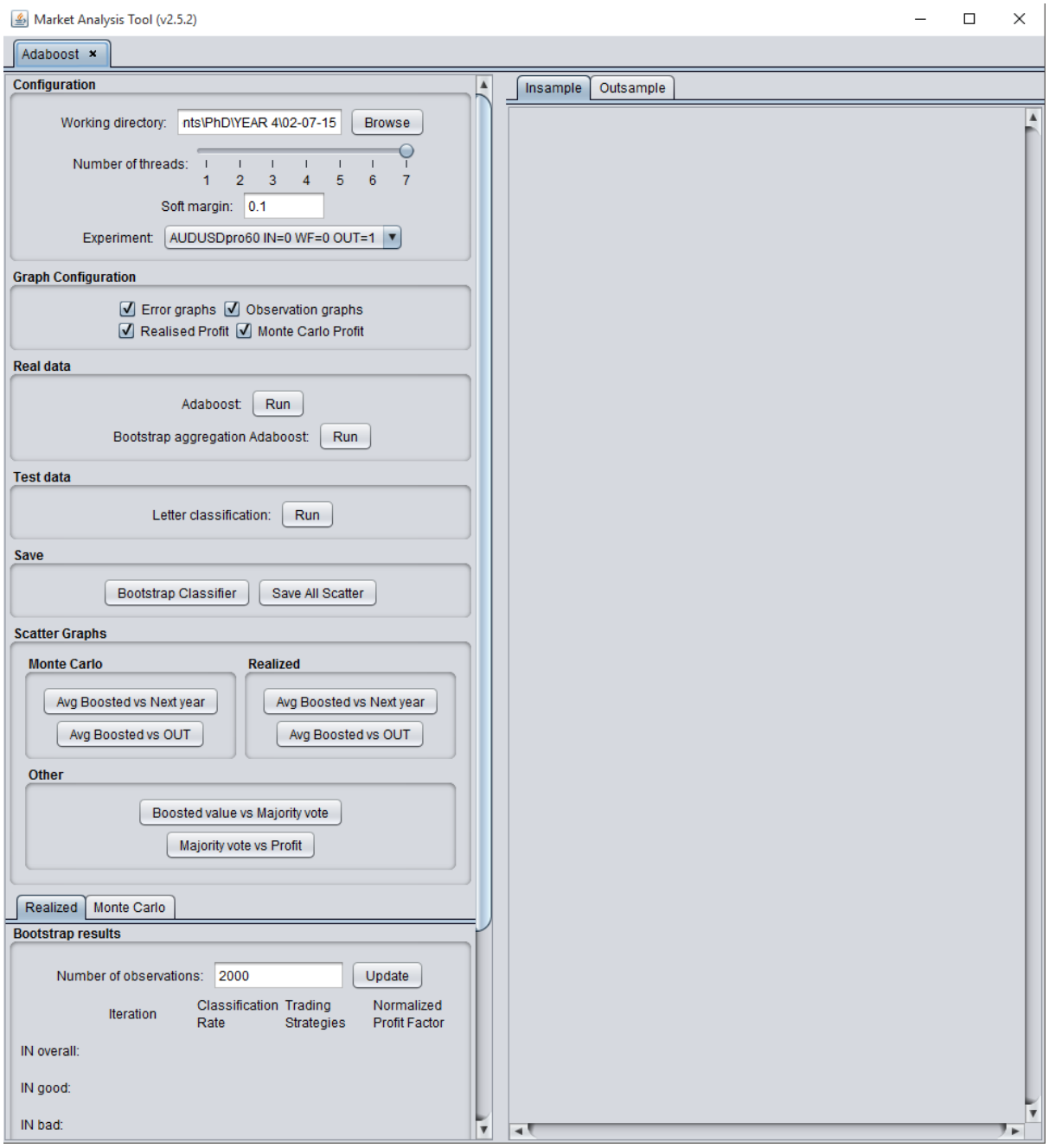


Figure 215: Overview of the boosting workspace

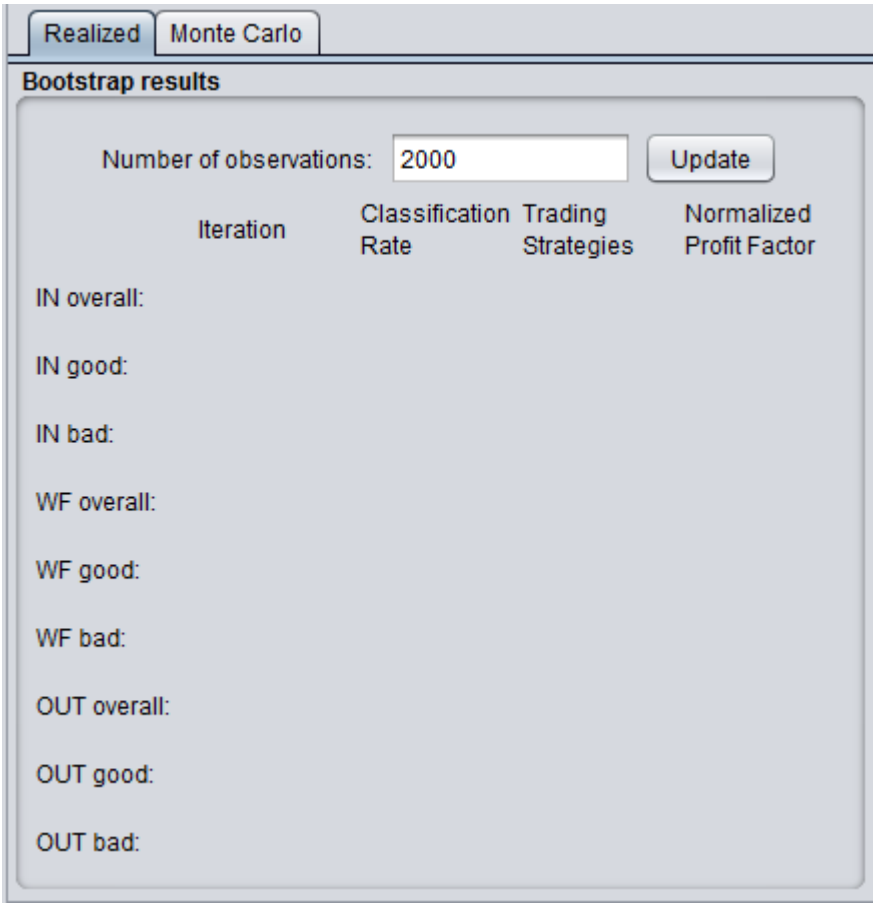


Figure 216: Overview of the boosting workspace continued

Technical Analysis Combiner Workspace

This part of the software visualises the search space of all possible trading strategies given a set of technical analysis algorithms (see Figure 217).

Main functionality:

- Visualise trading strategy search space
- Save the trading strategy scatter results

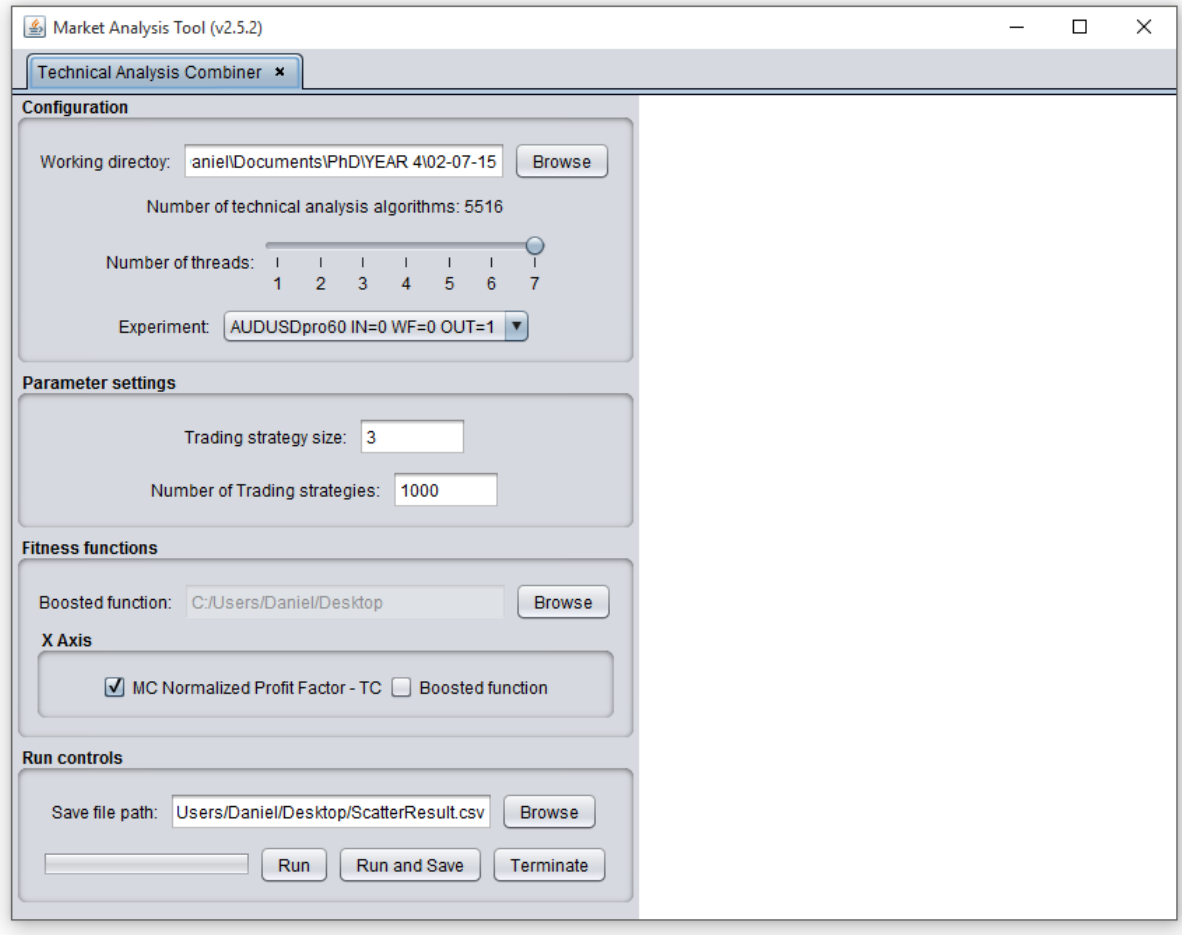


Figure 217: Overview of the technical analysis combiner workspace

Genetic Algorithm Workspace

This part of the software creates trading strategies by evolving random trading strategies using the Genetic Algorithm methodology (see Figure 218).

Main functionality:

- Shows mean, min, max and median fitness values of the population during each iteration of the Genetic Algorithm
- Allows the user to adjust the parameters of the Genetic Algorithm
- Allows the user to save the results to a file

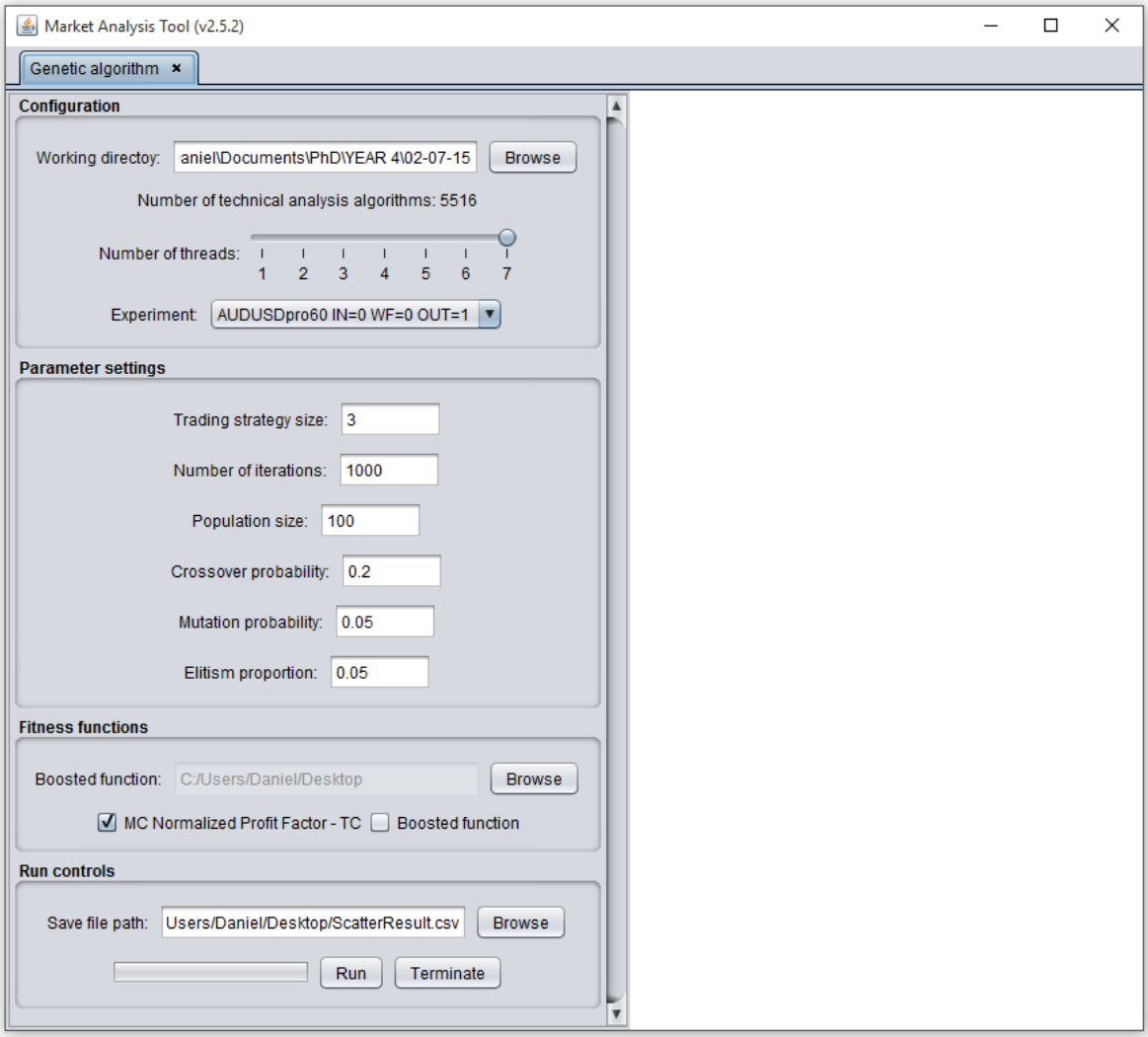


Figure 218: Overview of the Genetic Algorithm workspace

Live Trading Workspace

This part of the software interfaces with MetaTrader 4 (see Figure 219).

Main functionality:

1. Allows the user to trade trading strategies created in MaTool on MetaTrader 4. *This is done by using an Expert advisor in MetaTrader 4 which talks to the MaTool application*
2. Allows the user to configure the lot size and stop loss of the trading strategy
3. Allows the user to stop the trading strategy from trading
4. Allows the user to see the equity curve of the traded trading strategies

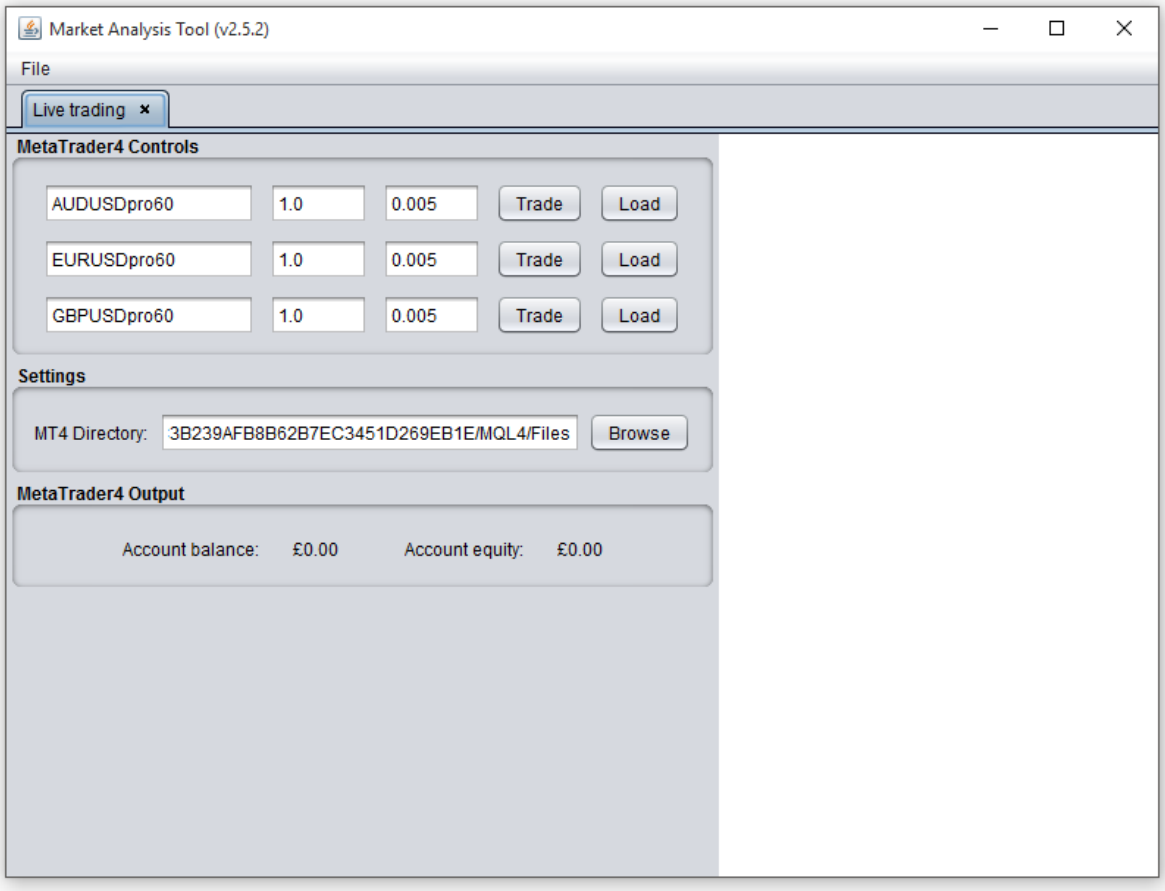


Figure 219: Overview of the live trading workspace

Portfolio Manager

This part of the software shows the relationship between the risk and reward of many markets from their equity curves (see Figure 220).

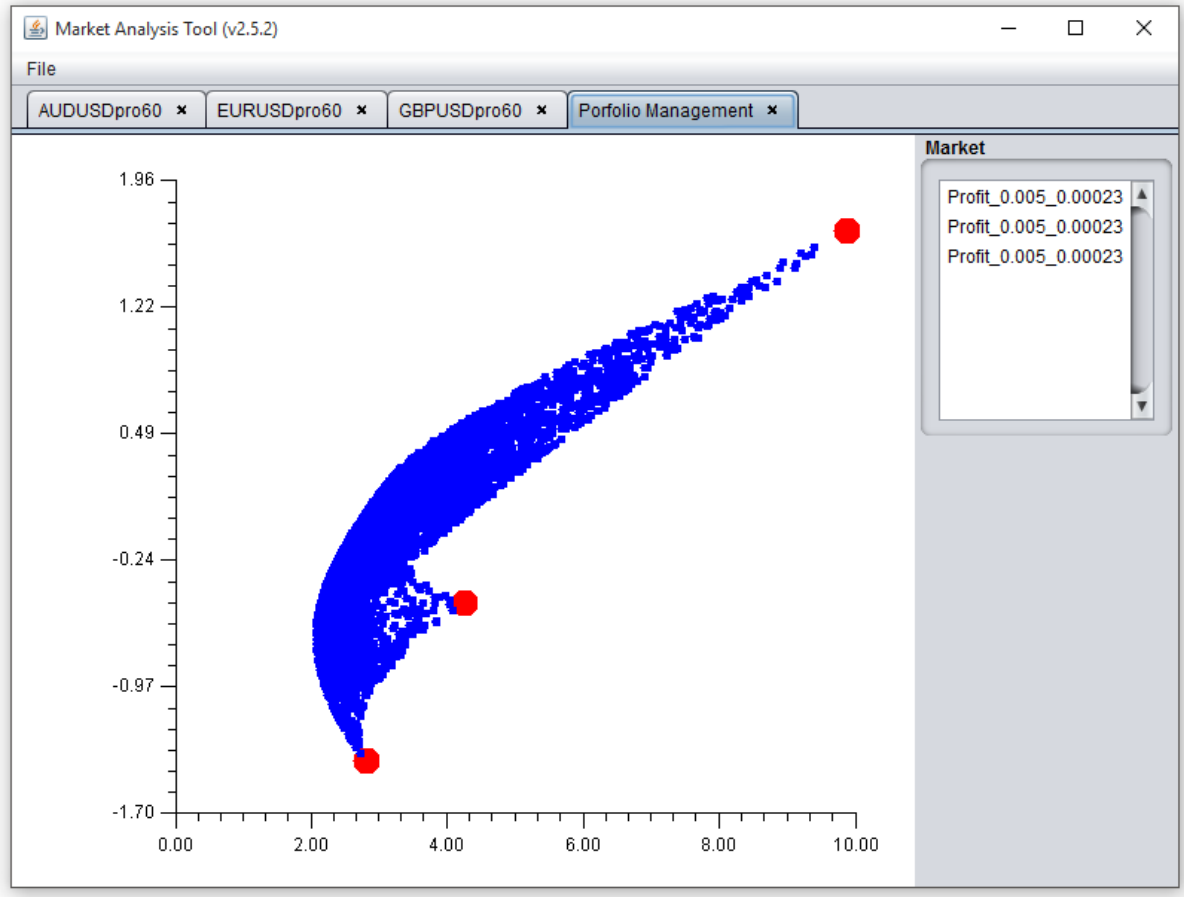


Figure 220: Overview of the portfolio manager workspace

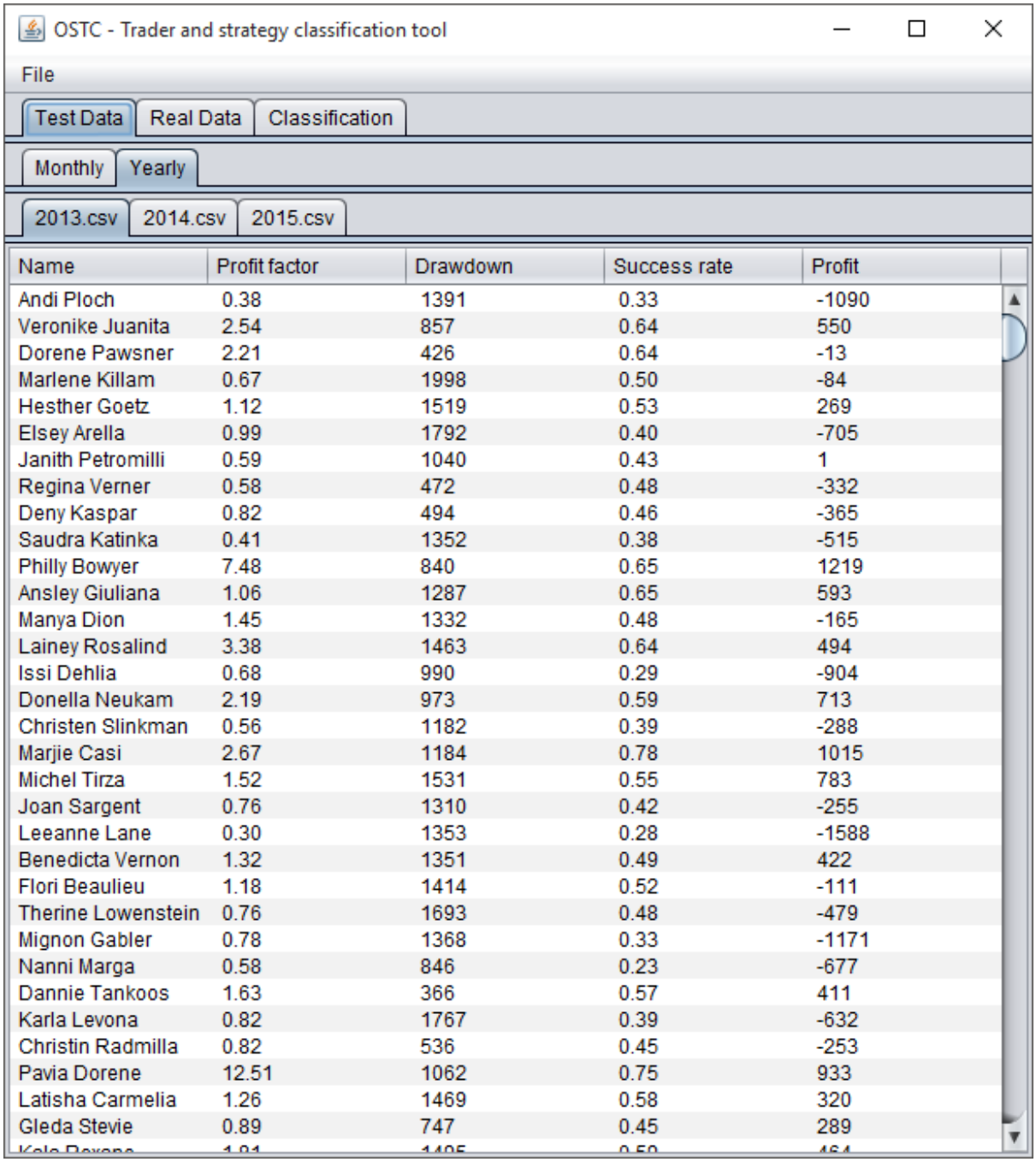
Appendix D – Trader Classification Tool

Chapters 8-11 explored the development of Adaboost classification systems for the classification of trading strategies. Trading strategies were used in this project as a proxy for actual trader data which, for reasons of commercial sensitivity, were not available. The project partner would benefit from having a program, with a usable interface, to appraise the performance of actual traders, using metrics of their trading performance.

The program described in this appendix is intended to assist managers to judge whether a trader is 'bad', and can indicate *tilting*. Tilt is a state of mental or emotional confusion or frustration where a trader deviates from their optimal trading system. A bad trade, problems outside of work and stress are a few things that lead to tilting. This program enables managers to help spot traders that are tilting, and early intervention would reduce the risk of losses of the company's capital. In addition to spotting 'bad' and tilting traders, this program can be used to help judge the amount company capital to invest into each trader as the performance of each trader can be scored and compared to other traders relatively.

This program can be used to classify and identify 'bad' trading strategies using the results in Chapter 11. Chapter 7 attempted to create potentially 'good' trading strategies, each of which contain a combination of technical analysis interpretations. The Genetic Algorithm suffered from early population convergence and later experiments used a random set of trading strategies. This program could benefit from including a 'potentially good' set of trading strategies using techniques described in the literature. The classification system may also benefit from a different trading strategy model, perhaps even trading strategies using artificial neural networks or support vector machines.

Figure 221 shows the user the raw performance metric data which is loaded into the program from the file system. The program loads the data from comma separated files (csv) in the folders "Test data" and "Real data" which are created relative to the program's current directory. The raw data is shown to ensure that the data is properly formatted



Name	Profit factor	Drawdown	Success rate	Profit
Andi Ploch	0.38	1391	0.33	-1090
Veronike Juanita	2.54	857	0.64	550
Dorene Pawsner	2.21	426	0.64	-13
Marlene Killam	0.67	1998	0.50	-84
Hesther Goetz	1.12	1519	0.53	269
Elsey Arella	0.99	1792	0.40	-705
Janith Petromilli	0.59	1040	0.43	1
Regina Verner	0.58	472	0.48	-332
Deny Kaspar	0.82	494	0.46	-365
Saudra Katinka	0.41	1352	0.38	-515
Philly Bowyer	7.48	840	0.65	1219
Ansley Giuliana	1.06	1287	0.65	593
Manya Dion	1.45	1332	0.48	-165
Lainey Rosalind	3.38	1463	0.64	494
Issi Dehlia	0.68	990	0.29	-904
Donella Neukam	2.19	973	0.59	713
Christen Slinkman	0.56	1182	0.39	-288
Marjie Casi	2.67	1184	0.78	1015
Michel Tirza	1.52	1531	0.55	783
Joan Sargent	0.76	1310	0.42	-255
Leeanne Lane	0.30	1353	0.28	-1588
Benedicta Vernon	1.32	1351	0.49	422
Flori Beaulieu	1.18	1414	0.52	-111
Therine Lowenstein	0.76	1693	0.48	-479
Mignon Gabler	0.78	1368	0.33	-1171
Nanni Marga	0.58	846	0.23	-677
Dannie Tankoos	1.63	366	0.57	411
Karla Levona	0.82	1767	0.39	-632
Christin Radmilla	0.82	536	0.45	-253
Pavia Dorene	12.51	1062	0.75	933
Latisha Carmelia	1.26	1469	0.58	320
Gleda Stevie	0.89	747	0.45	289

Figure 221: The interface for viewing raw data of traders or trading strategies

Figure 222 is a simplified interface which the user uses to classify traders and trading strategies. To create the classification system and then classify the traders or trading strategies in the outsample dataset, the user has to do the following:

- Choose which dataset to use. Use the test data directory or real data directory.
- Sort the data files from oldest to newest to ensure the validation set contains the newest dataset and the final file represents the more recent performance metrics and is used as the outsample file.
- Specify what proportion of the historical dataset is training data (used to create the classification system) and validation data.
- Specify the performance metric to classify and the value which separates the two classifications. For example, the user may want to separate traders into potentially profitable traders and potentially unprofitable traders. The user must specify the metric to classify as the profit factor metric and the decision boundary for the classification to be of the value of 1.

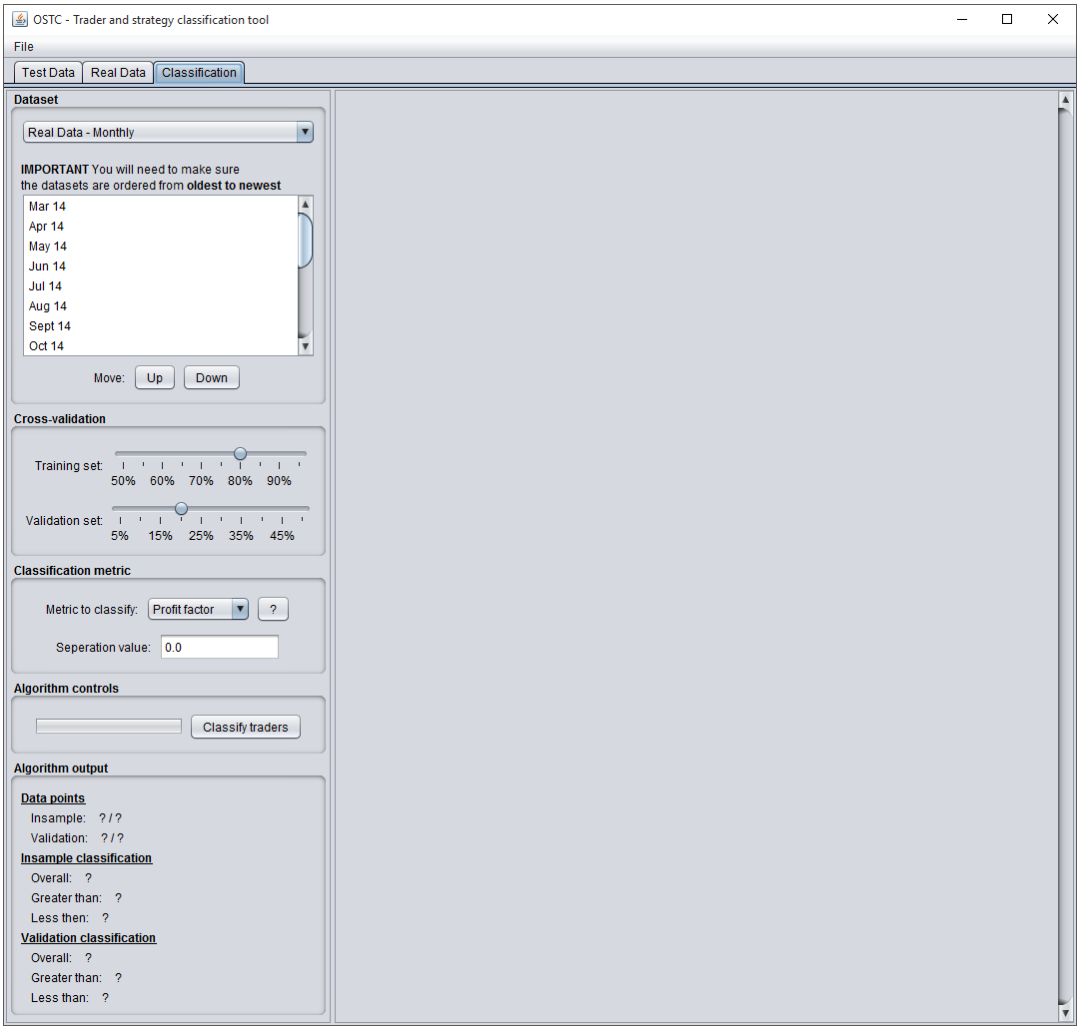
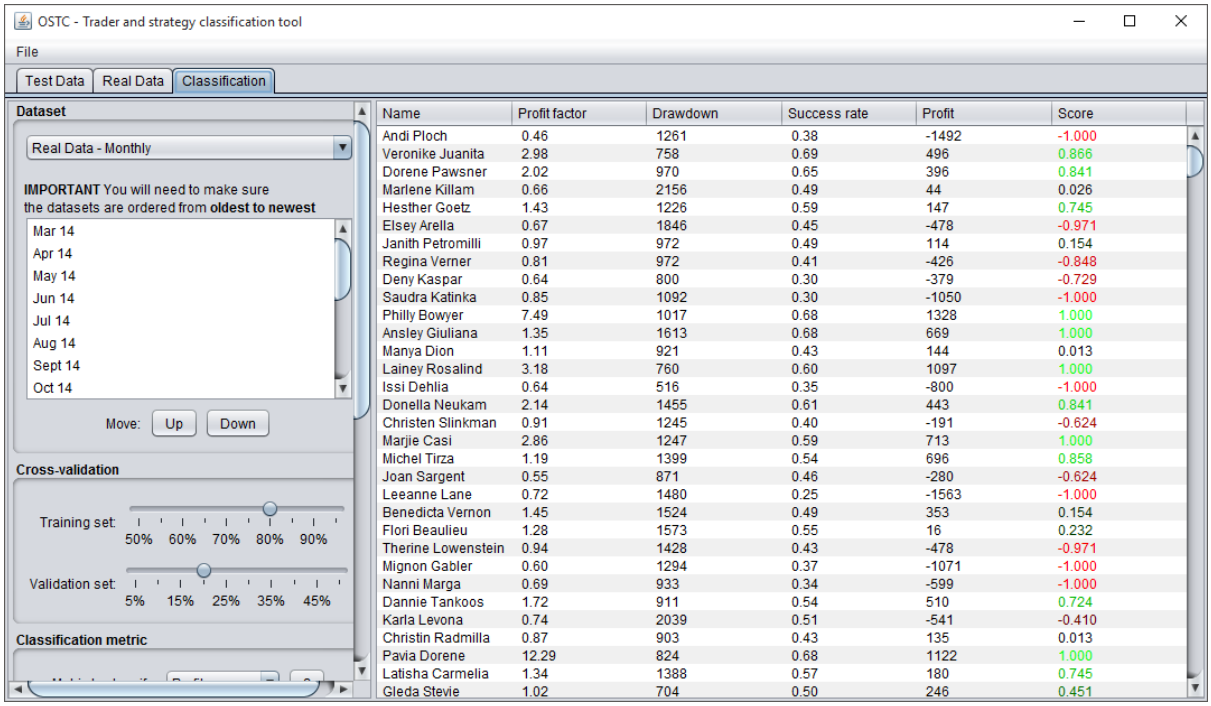


Figure 222: The interface which classifies traders and trading strategies

In this program, many parameter settings of the Adaboost based classification system using the bootstrap aggregation and feature bagging techniques are attempted. The resultant classification system with the highest overall classification rate in the validation set is used to classify the traders or trading strategies in the last dataset (the outsample dataset which attempts to predict future performance). The results of the final classification can be shown in the "Algorithm output" part of the program.

Figure 223 shows the classifier's score for each trader on the last data file provided by the user. The score is from -1 to 1 where -1 predicts below the classification decision boundary, 0 is the decision boundary and 1 is above the decision boundary. The closer the trader or trading strategy score is to -1 or 1.



Name	Profit factor	Drawdown	Success rate	Profit	Score
Andi Ploch	0.46	1261	0.38	-1492	-1.000
Veronike Juanita	2.98	758	0.69	496	0.866
Dorene Pawsner	2.02	970	0.65	396	0.841
Marlene Killam	0.66	2156	0.49	44	0.026
Hesther Goetz	1.43	1226	0.59	147	0.745
Elsey Arella	0.67	1846	0.45	-478	-0.971
Janith Petromilli	0.97	972	0.49	114	0.154
Regina Verner	0.81	972	0.41	-426	-0.848
Deny Kaspar	0.64	800	0.30	-379	-0.729
Saudra Katinka	0.85	1092	0.30	-1050	-1.000
Philly Bowyer	7.49	1017	0.68	1328	1.000
Ansley Giuliana	1.35	1613	0.68	669	1.000
Manya Dion	1.11	921	0.43	144	0.013
Lainey Rosalind	3.18	760	0.60	1097	1.000
Issi Dehlia	0.64	516	0.35	-800	-1.000
Donella Neukam	2.14	1455	0.61	443	0.841
Christen Slinkman	0.91	1245	0.40	-191	-0.624
Marjie Casi	2.86	1247	0.59	713	1.000
Michel Tirza	1.19	1399	0.54	696	0.858
Joan Sargent	0.55	871	0.46	-280	-0.624
Leeanne Lane	0.72	1480	0.25	-1563	-1.000
Benedicta Vernon	1.45	1524	0.49	353	0.154
Flori Beaulieu	1.28	1573	0.55	16	0.232
Therine Lowenstein	0.94	1428	0.43	-478	-0.971
Mignon Gabler	0.60	1294	0.37	-1071	-1.000
Nanni Marga	0.69	933	0.34	-599	-1.000
Dannie Tankoos	1.72	911	0.54	510	0.724
Karla Levona	0.74	2039	0.51	-541	-0.410
Christin Radmilla	0.87	903	0.43	135	0.013
Pavia Dorene	12.29	824	0.68	1122	1.000
Latisha Carmelia	1.34	1388	0.57	180	0.745
Gleda Stevie	1.02	704	0.50	246	0.451

Figure 223: Results of the program's classification on test data