

Automated Trading System for The Betfair Exchange

William Jack Alcock

Supervisor: Dr Martin Berger

Candidate Number: 105860

University of Sussex BSc Computer Science

April 18, 2016

Statement Of Originality

This report is submitted as part requirement for the degree BSc Computer Science at the University of Sussex. It is the product of my own labour except where specifically stated in the text. The report may be freely copied and distributed provided the source is acknowledged.

William Jack Alcock
April 18, 2016

Acknowledgements

I would like to acknowledge and thank

- Dr Martin Berger, University of Sussex for his encouragement, advice and support while supervising the project.
- Dr Novi Quadrianto, University of Sussex for his advice and help with machine learning models.
- Neil Thomas from Betfair and Alan Weber from Fracsoft for supplying historical Betfair market data
- Sue, Elizabeth, Poppy, Alex, Mum and Dad for their constant support and encouragement.

Summary

The gambling industry has been around for centuries but in recent years the nature of traditional betting markets have been revolutionised by the advent of electronic betting exchanges. Betfair is one of the leading betting exchanges. This project has two sections.

The first section aims to build a distributed trading system for the Betfair Exchange.

The second section aims to build a model to predict the direction of prices on the Betfair Exchange's horse racing markets by studying past market data using quantitative technical analysis and machine learning.

The dissertation covers the following areas:

- An introduction to the gambling industry and betting exchanges.
- Professional ethical considerations surrounding this project.
- Requirements analysis for both sections of the project.
- The design and implementation of a trading system for the Betfair Exchange.
- The implementation and testing of a model trained using pre-labelled historical market data to predict Betfair market trends.

Appendix [A](#) includes a glossary of gambling and machine learning terms used in this dissertation. Those unfamiliar with these subjects are advised to read this section first.

Contents

1	Statement Of Originality	1
2	Acknowledgements	2
3	Summary	3
	List of Figures	7
	List of Tables	8
4	Introduction	9
4.1	The Betfair Exchange	9
4.1.1	What is The Betfair Exchange	9
4.1.2	An Example of a Betfair Market	10
4.1.3	Trading a Betfair Market	10
4.2	Automated Trading Systems	12
4.3	Technical Analysis	13
4.4	Machine Learning	13
4.5	Project Objectives	14
4.5.1	Part 1	14
4.5.2	Part 2	14
5	Professional Considerations	16
5.1	Public Interest	16
5.2	Professional Competence and Integrity	16
5.3	Duty to Relevant Authority	17
5.4	Duty to Profession	17
6	Development Approach	18
6.1	Waterfall Vs. Agile	18
6.1.1	Waterfall	18
6.1.2	Agile	19
6.2	Testing	19
6.2.1	Test Driven Development	20
6.2.2	End-to-end vs Unit Tests	20
6.2.3	Unit Testing: State vs Behaviour	21
6.3	Version Control	21
6.4	Programming Languages	21
7	Requirements Analysis	23
7.0.1	Part 1 - Building a Trading System	23
7.0.2	Part 2 - Building a Predictive Model	25

8	The Trading System	26
8.1	Design	27
8.1.1	Client-Server Vs Peer-to-Peer Architecture	28
8.1.2	Client and WebServer	29
8.1.3	The Core Server	31
8.1.4	The Service Layer	34
8.2	Implementation	34
8.2.1	Central Configuration	35
8.2.2	Service Layer	36
8.2.3	Event Manager	39
8.2.4	Controller	40
8.2.5	Data Provider	40
8.2.6	Data Model	43
8.2.7	Web Server and Client	43
8.2.8	Navigation Data Service	44
8.2.9	Order Manager	45
8.2.10	Auto Trader	47
8.2.11	Data Store	49
9	Predicting Price Movement	50
9.1	The Data	50
9.2	The Indicators	51
9.3	The Algorithm	51
9.4	Decision Trees	52
9.5	Random Forest	53
9.5.1	Overhead of Training a Random Forest on Features with Continuous Values	54
9.6	Extremely Random Forest	54
9.7	Implementation	55
9.7.1	Labelling the Data	56
9.7.2	Reducing the Size of the Data	56
9.7.3	Adjustments for Unevenly Distributed Data	56
9.8	Optimisation	57
9.8.1	Out-Of-Bag Error	57
9.8.2	Optimisation Results	58
9.9	Results	59
10	Conclusion	60
10.1	Development Approach	61
	Appendices	62
A	Glossary of Terms	63
A.1	Gambling Terms	63
A.2	Machine Learning Terms	65
B	Controller API	67
C	Betfair Price Increments	68

D Historical Data Schema	69
E Technical Analysis Indicators	70
F Future Work	71
Bibliography	72

List of Figures

4.1	Annotated Betfair Market, Match odds: Swansea vs Arsenal	10
6.1	Waterfall Development Process	18
6.2	Agile Development Process	19
6.3	The Testing Pyramid	20
8.1	Trading System Architecture	27
8.2	Client-Server Architecture	28
8.3	Peer-to-Peer Architecture	28
8.4	GUI Main Window	30
8.5	GUI Ladder Window	31
8.6	Service Layer Message Flow	36
8.7	Test Service Layer Message Flow	37
8.8	Local Order Book Hierarchy of Objects	39
8.9	Market Book Price Projections	41
8.10	Market Book Market Data Limits	41
8.11	Data Provider Polling Mechanism	42
8.12	Web Server and Client Architecture	43
8.13	Navigation Data File Structure	45
8.14	Life Cycle of an Order	46
8.15	Basic Trading Strategy	47
9.1	Decision Tree for Mammal Classification	52
9.2	A Random Forest with 5 trees	53
9.3	Algorithm for growing a Decision Tree for an Extremely Random Forest[14]	55
9.4	Graphs of Optimisation Results	58
A.1	Example of underfitting, rightfitting and overfitting[9]	66

List of Tables

4.1	Scenario 1, Potential Profit and Liability	11
4.2	Scenario 2, Potential Profit and Liability	11
4.3	Scenario 3, Potential Profit and Liability	12
8.1	Number of markets/call by price projection	41
8.2	Price projection hierarchy	42
9.1	Implied Probability of Decimal Odds	51
9.2	Labelled Example Data	54
9.3	Training and Testing Datasets	56
9.4	Tables of Optimisation Results	58
9.5	Predicted Classes Vs Actual Classes	59
9.6	Distribution, Precision and Recall by Class	59
9.7	Weighted Accuracy, Precision and Recall	59

Introduction

This project has two parts.

1. Build a computer trading system for the Betfair Exchange that can be used to automate trading.
2. Use technical analysis and machine learning to train a model that can predict market trends.

The gambling industry has been around for centuries but in recent years the nature of traditional betting markets has been revolutionised with the advent of electronic betting exchanges. Betting exchange markets share some similarities with financial markets and the automated trading of financial markets is wide spread and a topical subject. The motivation behind this project was to see if some of the principles applied to financial trading can be applied to trade betting exchanges.

This section provides

- An introduction to the Betfair Exchange and how it can be traded.
- An explanation of the advantages of an automated trading system.
- A brief overview of technical analysis and machine learning.
- A list of the primary and secondary objectives for each part of the project.

4.1 The Betfair Exchange

4.1.1 What is The Betfair Exchange

Betfair was launched in June 2000 as the world's first betting exchange[2].




Traditionally High Street bookmakers offer you odds on an event happening and you only have the choice of backing the event at those odds or not.

The Betfair Exchange allows gamblers to choose the odds at which they want to back or lay an event. Betfair proposes the markets that can be bet on and matches peoples bets making a 5% (or less) commission on any winning bet. This is comparable to the way a financial exchange allows people to choose the price at which they buy and sell securities and matches their orders.

Betting exchanges generally have better odds than traditional bookmakers although it should be noted that commissions are not included in those odds.

Although subsequently other companies have launched betting exchanges, Betfair remains the biggest[38]. Today Betfair's Exchange processes over 1.2 billion bets a year, with a trading value of £56 billion; more transactions than all the major European Stock Exchanges combined[1].

4.1.2 An Example of a Betfair Market

3 selections		Back all		Lay all		
 Swansea	4.6 £2848	4.7 £4723	4.8 £1095	4.9 £1485	5 £3038	5.1 £2576
 Arsenal	1.81 £5397	1.82 £1367	1.83 £1239	1.84 £1594	1.85 £3590	1.86 £3135
 The Draw	3.9 £4016	3.95 £4276	4 £3642	4.1 £6103	4.2 £6831	4.3 £6430

Available to Back
Available to Lay

4.8
£1095

← Price

← Available size

Figure 4.1: Annotated Betfair Market, Match odds: Swansea vs Arsenal

Figure 4.1 shows an example of a Betfair market for the match odds of a football match between Swansea and Arsenal.

The market is split into three selections, one for each possible outcome:

- Swansea winning.
- Arsenal winning.
- A draw.

The odds for each outcome are split into two sides:

- Prices available to Back - because of unmatched Lay bets in the market.
- Prices available to Lay - because of unmatched Back bets in the market.

Displayed underneath each price is the total size (or volume) of all the unmatched bets at that price. The best available prices to back and lay are coloured blue and pink respectively.

A punter looking to Back a selection would want the highest possible price to maximise the return on their stake. A punter looking to Lay a selection would want the lowest possible price to minimise liability. The selection's current price is where the price of the unmatched Back and Lay bets meet.

4.1.3 Trading a Betfair Market

Just as prices on financial markets fluctuate over time so will the prices on Betfair's markets.

A trader can profit by laying a selection when the price is low and backing when the price is high or visa-versa. To further explain this consider the following scenarios:

Scenario 1

A trader has backed Swansea to win for £10 at odds of 5.0 before the match starts. By half time the score is 2-0 to Swansea and the available price to back is now 1.49 and to lay 1.5 because the probability of Swansea winning is now much higher than it was before the match started. The trader now makes a lay bet for the selection at the lower price of 1.5.

Bet	Size	Price	Profit	Liability
Back	£10.00	5.00	£40.00	£10.00
Lay	£10.00	1.50	£10.00	£ 5.00

Table 4.1: Scenario 1, Potential Profit and Liability

Table 4.1 shows the potential profit and liability from placing these bets.

- If Swansea win the trader will profit £40.00 from the back bet but will be liable to a £5.00 loss from the lay bet, leaving an overall return of £35.00.
- If Swansea lose the trader will profit £10.00 from the lay bet but will be liable to a £10.00 loss from the back bet, leaving an overall return of £0.00.

By placing both bets the trader has ensured that whatever the outcome they will not lose any money.

Scenario 2

Bet	Size	Price	Profit	Liability
Back	£10.00	5.00	£40.00	£10.00
Lay	£40.00	1.50	£40.00	£20.00

Table 4.2: Scenario 2, Potential Profit and Liability

A trader has backed Swansea to win before the match as in scenario 1 but this time the trader places a larger lay bet of £40.00 at 1.5.

Table 4.2 shows the potential profit and liability from placing these bets.

- If Swansea win the trader will profit £40.00 from the back bet but will be liable to a £20.00 loss from the lay bet, leaving an overall return of £20.00.
- If Swansea lose the trader will profit £40.00 from the lay bet but will be liable to a £10.00 loss from the back bet, leaving an overall return of £30.00.

In this scenario the trader has ensured that whatever the outcome they are guaranteed a return, £20.00 if Swansea win and £30.00 if Swansea lose. This process is known in gambling terms as "cashing out" or "greening up".

Scenario 3

Bet	Size	Price	Profit	Liability
Back	£10.00	5.00	£40.00	£10.00
Lay	£ 3.00	16.00	£ 3.00	£45.00

Table 4.3: Scenario 3, Potential Profit and Liability

A trader has backed Swansea to win for £10.00 at odds of 5.0 before the match starts but in this scenario at half time the score is 2-0 to Arsenal and the available price to back and lay Swansea is much higher at 15.00 to back and 16.00 to lay because the probability of Swansea winning is now lower. The trader now makes a lay bet of £3.00 at 16.00 to reduce the size of their potential loss.

Table 4.3 shows the potential profit and liability from placing these bets.

- If Swansea win the trader will profit £40.00 from the back bet but will be liable to a £45.00 loss from the lay bet, leaving an overall loss of £5.00.
- If Swansea lose the trader will profit £3.00 from the lay bet but will be liable to a £10.00 loss from the back bet, leaving an overall loss of £7.00.

In this scenario the trader has minimised the potential loss from the initial back bet, this process is known as stopping out of a trade.

4.2 Automated Trading Systems

Automated trading systems are widely used in financial markets and allow the trader to establish specific rules for:

- Which markets to trade.
- The price and type of trades to place in a market.

Automating the trading of markets offers many benefits compared to manual trading including:

- Minimising emotions - a trader maybe affected by emotions and disregard their rules either because they are unsure about entering a trade or because they are reluctant to exit one.
- Achieve consistency - if a trader does not follow their rules it is hard to compare progress over time.
- Improve order entry speed - computers may respond to changing market conditions and place and cancel orders faster than a human.
- Trade a wider range of markets - computers can process information and trade over a wider range of markets simultaneously compared to a human.
- Ability to Backtest - an automated trading strategy can be tested on historical markets to see how well it would have performed, providing an indication of future performance.

To aid automation of trading on Betfair's markets the trader should be able to predict the outcome of an event or the direction in which the price will move.

Both technical and fundamental analysis are used to predict the future price movement of financial markets. Technical analysis concentrates on the study of market action by looking at the previous price movement of a security. Fundamental analysis focuses on the economic forces of supply and demand that cause prices to move higher, lower, or stay the same[27].

When applied to a sporting event such as a horse race fundamental analysis would look at factors such as:

- How the horse performed previously.
- How the jockey performed previously.
- The distance of the race.
- The condition of the race course.

In contrast technical analysis looks at how the price and betting activity in the market had evolved and how that compared to previous markets.

4.3 Technical Analysis

Technical Analysis assumes that the forces of supply and demand are already incorporated in the market price. A higher volume of unmatched bets in the market to back a horse as opposed to lay will drive the price down as traders place back bets at lower prices in an effort to get matched. The underlying premise is that if the price of a selection is moving lower then the trader should back the selection with a view to laying it at a lower price, making a profit from following the markets trend.

Quantitative technical analysis is the application of mathematical indicators on previous price data to predict a future trend. They are commonly based on the high, low and closing price and volume traded over intervals of equal duration in the market data. A simple example could be a moving average of closing prices at 1 minute intervals since a market opened. If the current price is above the moving average it could be predicted to rise, if it is below the moving average it could be predicted to fall. Studies have shown that quantitative technical analysis can be effective when used by an automated agent to predict market direction on financial markets[40].

In *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*[27] John Murphy states that 'One of the great strengths of technical analysis is its adaptability to virtually any trading medium and time dimension'.

The second part of this project will use quantitative technical analysis and machine learning to predict the future direction of a market from historical price data.

4.4 Machine Learning

Machine learning is the science of getting computers to act without being explicitly programmed[28]. Machine learning focuses on the development of algorithms that can learn from patterns in data

to build a model that can make predictions. These algorithms aim to build a model that encodes a mapping from a set of features in the data to an output and can be split into two categories:

- Supervised learning - The algorithm is given examples of the data that have already been labelled with an output. This training data is used to build a model that encodes a mapping from the data to the output. Supervised learning algorithms can be divided into two types
 - Classification - which maps the data into a distinct group.
 - Regression - which maps the data into a real value.
- Unsupervised learning - The algorithm is given examples that are not labelled and uses them to find structure in the data e.g; Clustering, which groups the data based on its features.

4.5 Project Objectives

This project has two parts, to build a trading system for the Betfair Exchange and to train a model that can be used to predict price trends on Betfair markets.

4.5.1 Part 1

The primary objectives are to build a trading system that can be used to:

- Navigate markets on the Betfair Exchange.
- Manually trade markets on the Betfair Exchange.
- Automate the trading of markets on the Betfair Exchange.

The secondary objectives are to add functionality to:

- Record market data and trading activity to be analysed after the fact.
- Provide a framework to back-test an automated trading strategy against recorded market data.

The trading system should include a graphical user interface (GUI). The GUI design however is not the focus of this project and the emphasis will be on adding functionality to the trading system rather than the design aspects of the user interface.

This part of the project will not be concerned with the resiliency or security of the trading system. These are complex subjects and outside the scope of this project.

4.5.2 Part 2

The primary objectives are to:

- Select a machine learning algorithm to be used to build a model from historical market data.
- Process the historical market data into intervals and select and apply technical analysis indicators.

- Divide the data into sets that can be used to train and test the model.
- Optimise the model using the training data.
- Analyse the results of the model on the testing data.

The secondary objectives are to:

- Integrate the model into a trading strategy.
- Automate the trading strategy using the system built in the first part of this project.

The primary objectives of both parts of this project are achievable within the time limits. The secondary objectives will only be met if there is enough time after the primary objectives have been met.

Professional Considerations

To ensure the integrity of this project it is important to comply with the British Society of Computing (BSC) Code of Conduct. Each of the four sections of the BSC Code of Conduct will be discussed:

5.1 Public Interest

From the Code of Conduct: [\[23\]](#)

You shall:

- *Have due regard for public health, privacy, security and well being of others and the environment.*
- *Have due regard for the legitimate rights of Third Parties*.*
- *Conduct your professional activities without discrimination on the grounds of sex, sexual orientation, marital status, nationality, colour, race, ethnic origin, religion, age or disability, or of any other condition or requirement*
- *Promote equal access to the benefits of IT and seek to promote the inclusion of all sectors in society wherever opportunities arise.*

This does not apply to this project as no user information will be collected and the project does not involve interaction with any Third Parties

5.2 Professional Competence and Integrity

From the Code of Conduct: [\[23\]](#)

You shall:

- *Only undertake to do work or provide a service that is within your professional competence.*
- *NOT claim any level of competence that you do not possess.*
- *Develop your professional knowledge, skills and competence on a continuing basis, maintaining awareness of technological developments, procedures, and standards that are relevant to your field.*
- *Ensure that you have the knowledge and understanding of Legislation* and that you comply with such Legislation, in carrying out your professional responsibilities.*
- *Respect and value alternative viewpoints and, seek, accept and offer honest criticisms of work.*
- *Avoid injuring others, their property, reputation, or employment by false or malicious or negligent action or inaction.*

- *Reject and will not make any offer of bribery or unethical inducement.*

This project will adhere to this section of the Code of Conduct and the author has been assessed to ensure they have the correct level of competency to undertake the project. Once the project has been completed the author will meet with the relevant supervisors to discuss the decisions taken during the project.

5.3 Duty to Relevant Authority

From the Code of Conduct: [\[23\]](#)

You shall:

- *Carry out your professional responsibilities with due care and diligence in accordance with the Relevant Authoritys requirements whilst exercising your professional judgement at all times.*
- *Seek to avoid any situation that may give rise to a conflict of interest between you and your Relevant Authority.*
- *Accept professional responsibility for your work and for the work of colleagues who are defined in a given context as working under your supervision.*
- *NOT disclose or authorise to be disclosed, or use for personal gain or to benefit a third party, confidential information except with the permission of your Relevant Authority, or as required by Legislation*
- *NOT misrepresent or withhold information on the performance of products, systems or services (unless lawfully bound by a duty of confidentiality not to disclose such information), or take advantage of the lack of relevant knowledge or inexperience of others.*

The author will adhere to this section of the Code of Conduct.

5.4 Duty to Profession

From the Code of Conduct: [\[23\]](#)

You shall:

- *Accept your personal duty to uphold the reputation of the profession and not take any action which could bring the profession into disrepute.*
- *Seek to improve professional standards through participation in their development, use and enforcement.*
- *Uphold the reputation and good standing of BCS, the Chartered Institute for IT.*
- *Act with integrity and respect in your professional relationships with all members of BCS and with members of other professions with whom you work in a professional capacity.*
- *Notify BCS if convicted of a criminal offence or upon becoming bankrupt or disqualified as a Company Director and in each case give details of the relevant jurisdiction.*
- *Encourage and support fellow members in their professional development*

The author will adhere to this section of the Code of Conduct.

Development Approach

6.1 Waterfall Vs. Agile

6.1.1 Waterfall

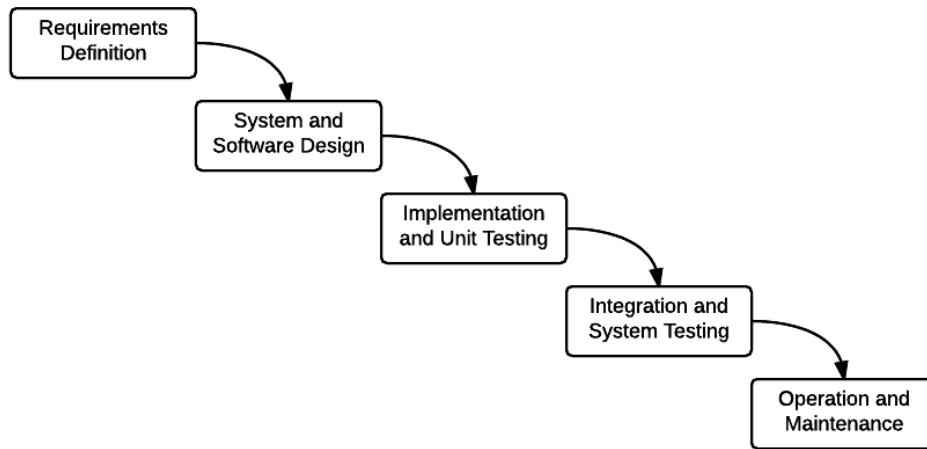


Figure 6.1: Waterfall Development Process

Figure 6.1 shows the stages of the Waterfall model for software development. In the Waterfall model each stage feeds down into the next one. Waterfall development employs a sequential design process from start to end and requires intensive planning and documentation, specifying that all aspects of the system are designed upfront before they are implemented with the testing coming last.

The advantages of the Waterfall model are:

- Easy forward planning and implementation.
- Easier to estimate and adhere to deadlines, budgets and rigid customer specifications.
- Ability to see and communicate the projects status.
- Better documentation of each stage

The disadvantages of the Waterfall model are:

- Inflexible partitioning of each stage makes it hard to incorporate changes during a project.
- No working product until the end of the project.
- Any dependency delays can halt development.

6.1.2 Agile

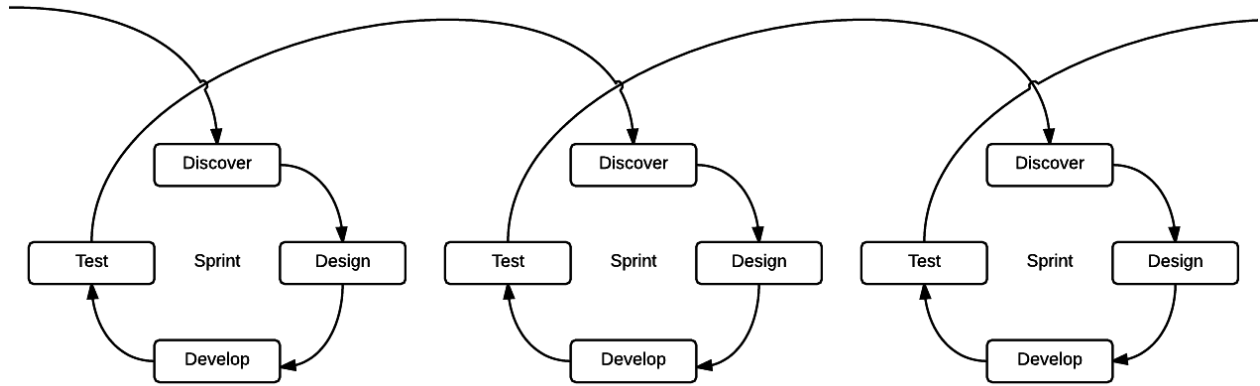


Figure 6.2: Agile Development Process

The Agile method proposes an incremental and iterative approach to software development. The development process is broken into sprints as shown in Figure 6.2, each sprint focusing on a small module of the system. Once each module is completed and tested it is integrated into the rest of the system and all system tests run to ensure the existing functionality has not been affected.

The advantages of the Agile model are:

- Projects benefit from the flexibility and ability to quickly incorporate changes.
- Continuous improvement as lessons are learnt from each stage.
- There is a working version of the system at the end of each sprint.

The disadvantages of the Agile model are:

- Its more difficult to predict the cost and delivery date upfront.
- Large projects can become unmanageable if the development life cycle is unclear at the beginning.
- It is harder to co-ordinate changes with large teams working over different locations.

This project would be best suited to an Agile development process because:

- It is relatively small with only one developer working on it.
- It is the first time the developer has written such a system and not all of the requirements of the system are known at the initial stage.
- There will always be a working version of the system from the last iteration reducing the risk of not having a working system when development comes to an end.

6.2 Testing

The Agile process places an emphasis on continuous testing, this section outlines the projects approach to testing.

6.2.1 Test Driven Development

Test Driven Development (TDD) stipulates that the unit tests for each module are written before the code. This requires giving consideration to the functionality and desired output of the code before it is written. The tests will fail before the code is written and pass once the code has met the requirements.

This project will use TDD where possible but because the developer has not written a trading system before, or used some of the technologies involved before, part of the development will be more exploratory. In these cases tests will be written after each module has been completed. The disadvantage of this approach is that during testing the code may need re-factoring so that it can be more easily tested.

The advantages of testing is:

- It proves the output of a given piece of code given a deterministic input.
- It allows the programmer to re-factor code with confidence existing functionality has not been broken.
- It helps document code - looking at the tests gives a clearer indication of what the code is doing.
- It encourages the programmer to write modular code.

6.2.2 End-to-end vs Unit Tests

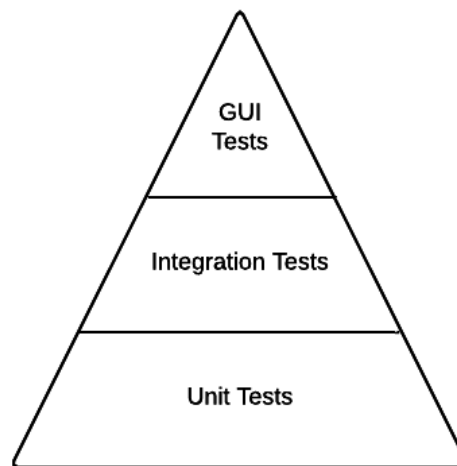


Figure 6.3: The Testing Pyramid

The Testing Pyramid shown in Figure 6.3 was developed by Mike Cohn in the book 'Succeeding with Agile'[26]. The pyramid shows that there should be more low-level unit tests than high-level GUI (end-to-end) tests.

End-to-end tests are harder to write and maintain because each one covers a larger area of the system and need to be changed every time the system does whereas each unit test only concerns

a small piece of the code. Other external pieces of code are replaced in the test with mocks. The advantage of this approach is it is easy:

- To maintain the code as only the tests for the unit of code being modified will need changing.
- To extend the software as new units of code and tests can be added without affecting existing ones.
- To identify which unit is failing as the failing test will identify the relevant unit of code.

When an end-to-end test fails its hard to identify the piece of code responsible because the test covers a wider part of the code.

For these reasons the project will focus on writing unit tests and some integration tests to ensure modules communicate correctly.

6.2.3 Unit Testing: State vs Behaviour

Unit tests can be split into two types:

- State tests - assert that given a deterministic input a unit of code returns a deterministic output
- Behaviour tests - given a deterministic input a unit of code makes the correct calls to external parts of code. The behaviour is verified by replacing the external parts of code with mocks and making assertions on when they are called and with what parameters.

This project uses a combination of state and behaviour tests. The implementation section includes a description of how each part of the system was tested.

6.3 Version Control

Version control is a system that records changes to a file or a set of files over time so that you can recall specific versions later[16]. Changes to a file or set of files can be grouped as commits and pushed to a central repository. The central repository provides a backup of the files and allows multiple developers access to the same code facilitating collaboration.

To support the iterative Agile development process this project will use GIT[15] version control software and GitHub to host the central repository. GIT is free, reliable, well documented, widely used in industry and has a large support community on the internet. Although this project has only one developer hosting the project on GitHub will aid in providing supervisors with reports on the projects progress.

6.4 Programming Languages

The majority of the project will be written in Scala[25]. The main reasons for choosing Scala above other languages are:

- The support for both functional and object orientated programming styles will increase development speed.

- Scala code will run across multiple operating systems as it compiles to Java Byte Code and runs on the Java Virtual Machine (JVM).
- There are a wide range of mature libraries available for Scala and it provides access to all existing Java libraries because it runs on the JVM.
- Scala encourages the use of immutable data types providing an advantage when writing multi-threaded code.
- Because Scala is a statically typed language errors can be caught at compile time.

Tests will be written using ScalaTest and ScalaMock. The Scala Build Tool (SBT) will be used to manage dependencies and builds and the code will be written using IntelliJ IDEA [\[3\]](#)

Requirements Analysis

This section lists brief functional (what the system shall do) and non-functional (how the system shall do it) requirements for each part of the project. Because this is an Agile project this is not an exhaustive list and new requirements will be incorporated as they are discovered during the development process.

7.0.1 Part 1 - Building a Trading System

The requirements were formed by considering the requirements of a trader and the constraints imposed by the Betfair Exchange's API[31].

The requirements of a trader are that they must be able to:

- Navigate markets on the exchange.
- Receive real time updates for markets on the exchange.
- Place and cancel orders across all markets on the exchange.
- Receive real time updates on the status of their orders on the exchange.
- Record data for selected markets on the exchange.
- Test a trading strategy without placing orders on the exchange.
- Action an automated strategy on markets on the exchange.

The constraints imposed by the Betfair Exchange's API are that:

- Navigation data for applications must be retrieved in the form of a compressed file[37].
- The exchange must be polled at regular intervals for market data updates.
- The exchange must be polled at regular intervals for order updates.
- The number of requests for a single market's data must not exceed 5 per second[34].
- The amount of data requested in each call to the exchange must be within a maximum weighting[36].
- Any party making calls to the exchange must first authenticate[35].
- All subsequent calls to the exchange must contain a session token and application key[30].
- The exchange provides a service with delayed data for testing but does not allow betting transactions.

As a result

The functional requirements are the system must:

- Poll the exchange for data on selected markets and notify the user when the data changes.
- Place and cancel orders at the exchange.
- Poll the exchange for data on the markets with unmatched orders and notify the user when their status changes.
- Provide the user with a means to navigate markets on the exchange.
- Let the user record data for selected markets.
- On request action an automated trading strategy on selected markets.
- Let the user test an automated strategy on selected markets.

The non-functional requirements are the system must:

- Download Navigation data on start-up.
- Authenticate to the exchange on start-up.
- Store the session token and application key and apply them to all exchange calls.
- Poll the exchange for updates at regular intervals not exceeding 5 calls per second per market.
- Group calls to the exchange so that data request limits are not exceeded.
- Implement a local order book to simulate the placement of orders in a test environment.
- Make and receive calls asynchronously without blocking.

7.0.2 Part 2 - Building a Predictive Model

Betfair and their third party data provider Fracsoft supplied one months market data for horse racing markets from the 11th October 2014 to the 10th November 2014 showing sub-second deltas in the market. The requirements were formed by considering the requirements of a trader from a predictive model and by looking at the constraints imposed by the historical data.

The requirements of a trader from a predictive model are that it must:

- Produce accurate results
- Run in real time

The constraints imposed by the historical Betfair Exchange data are it must:

- Be separated by market (Race) and selection (Runner)
- Be grouped into intervals so technical analysis indicators can be applied
- Transformed into a format where it can be sorted and rearranged

As a result the functional requirement of the model is that it make accurate predictions and the non-functional that it return predictions in real time.

The Trading System

This section of the report will start by describing the design of the trading system in its current state and then cover the implementation of each part of the system in the order they were built.

8.1 Design

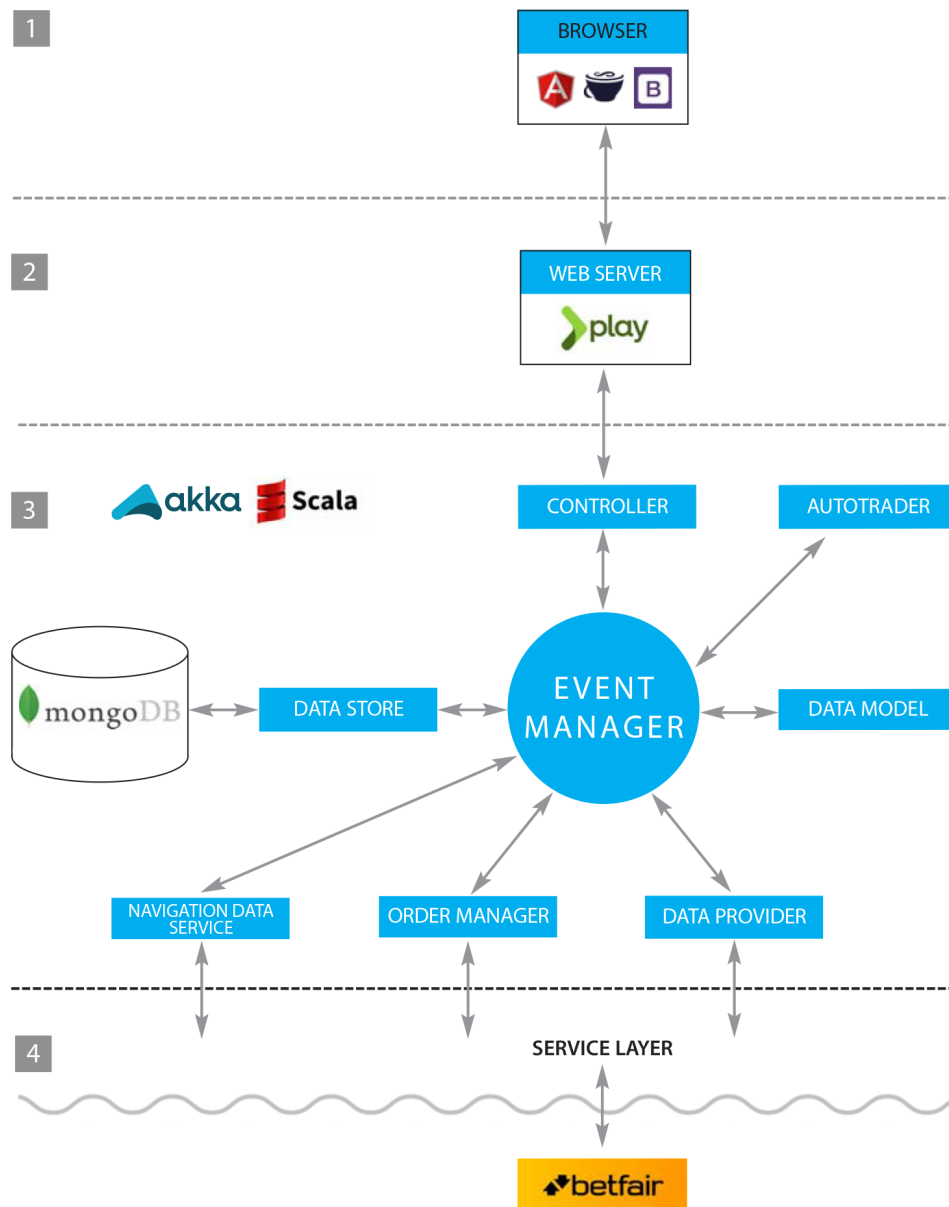


Figure 8.1: Trading System Architecture

The trading system implements a distributed Client-Server Architecture shown in Figure 8.1 and detailed below.

1. The Client is a thin client, running in a web browser and provides a graphical user interface (GUI).
2. The client pages are served by a web-server which facilitates communication from the client to the core-server through the core-server's controller.
3. The Core Server implements an event driven architecture managed by a central event manager.

4. The Core Server communicates with the Betfair Exchange through a service layer. The service layer is responsible for making Http calls to the Betfair Exchanges API and abstracting those calls into a simpler API which it exposes to the Core-Server.

8.1.1 Client-Server Vs Peer-to-Peer Architecture

A Client-Server Architecture, shown in Figure 8.2, separates the client from the server. The Client and the server communicate over a wire protocol which is kept to a minimum. The Client sends messages called requests to the Server, which processes these messages and returns requested information to the Client.

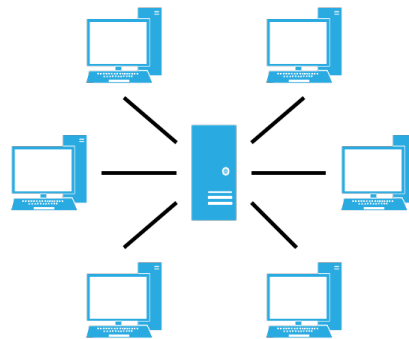


Figure 8.2: Client-Server Architecture

The most common alternative to a Client-Server Architecture is a Peer-to-Peer Architecture, shown in Figure 8.3. A Peer-to-Peer Architecture distinguishes itself by its distribution of power and function. In a Peer-to-Peer Architecture all the nodes have the same capabilities and responsibilities and communicate with each other rather than with a server.

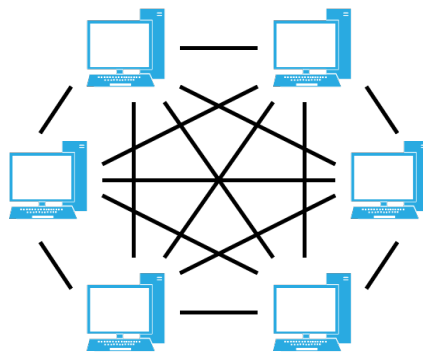


Figure 8.3: Peer-to-Peer Architecture

A Client-Server Architecture was chosen because:

- It enforces modularity - because the Client and Server are connected through a small wire protocol they can be developed in isolation as long as each one adheres to the protocol. The implementation for either the Server or Client can be changed without affecting the other as long as the wire protocol is adhered to.

- The server has central control of the data and resources and administers their access by the client. Although there is a downside to that as the server is a central point of failure, this project focuses on adding functionality opposed to ensuring resilience.
- Scalability - New changes, services and resources can be integrated with ease by upgrading the server, Peer-to-Peer Architecture does not scale well as the number of nodes increase because all the nodes need to be connected.
- Because the majority of processing happens on the server client machines don't need the same processor speed and resources as the Server.

8.1.2 Client and WebServer

Thin Client Vs. Thick Client

A thick client is one that will perform the bulk of the processing in a client-server application and is less dependent on communication with the server.

- A thick client requires lower server resources but the client needs to have more processing power and resources to run.
- Clients may be able to work off-line only requiring intermittent communication with the server.
- It is harder to deploy and make changes to a system that implements a thick client as changes need to be duplicated across all clients.

A thin client is designed to be especially small so that the bulk of the processing occurs on the server.

- A thin client is easier to deploy requiring little installation.
- A system with a thin client is easier to change because the majority of processing logic is on the server.
- A thin client is reliant on constant communication from the server, if the server is down the client won't work.
- A thin client requires less resources from the machine it is running on so a wider range of devices can be used as clients.

It was decided this system implement a thin client because:

- The processing logic does not have to be duplicated on both the client and server.
- The client needs to receive constant updates from the exchange and therefore lends itself the design.
- Its easy to deploy and change the client.

Implementing the client in a web browser lends itself to this design because no code needs to be deployed on the clients' devices, instead it is sent dynamically, page at a time to the client on request. If the client needs to be changed the change only needs to be made once on the Web-Server. If the number of clients needs to be scaled up then the Web-Server can be assigned extra resources. Another advantage of a client implemented in a web browser is that it is platform independent and can run on many different operating systems.

A more in-depth explanation of how the communication between the client, Web-Server and Core-Server works is included in the implementation section.

GUI Visual Design

This project does not focus on visual design. To increase development speed the styling has been based on Betfair's website. The GUI currently consists of two pages.



Figure 8.4: GUI Main Window

The GUI Main window shown in Figure 8.4 is split into three sections:

- The Navigation Window - lets the user navigate events and markets on the exchange and select the content of the detail window.
- The Detail Window - shows high level price data for events and markets selected using the navigation window.
- The Position Window - shows the user's unmatched bets and matched bets grouped by market and selection.

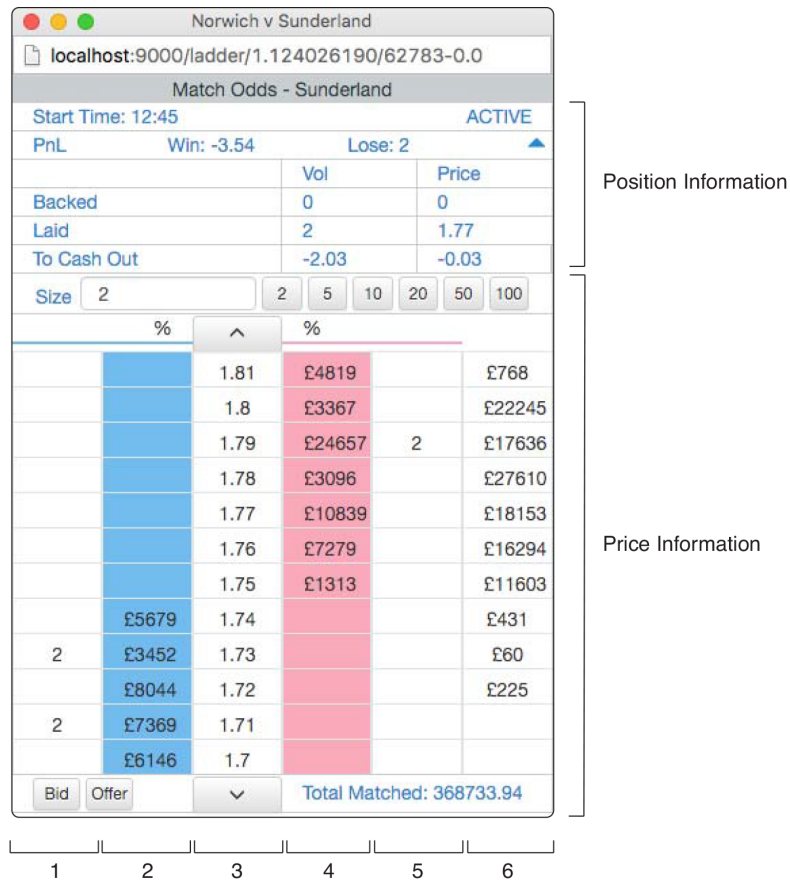


Figure 8.5: GUI Ladder Window

The GUI Ladder Window shown in Figure 8.5 lets the user navigate the full price depth of a selection. The upper section of the window shows information about the user's position. The lower section shows price information, each row relates to a different price, with the columns showing the following information in regard to that price:

1. The size of the users unmatched lay bets at the exchange.
2. The size of all the unmatched lay bets (or the size available to back) at the exchange.
3. The price
4. The size of all the unmatched back bets (or the size available to lay) at the exchange.
5. The size of the users unmatched back bets at the exchange.
6. The total volume trader since the market opened.

8.1.3 The Core Server

Event Based Architecture

An Event Based Architecture (EDA) is a framework that orchestrates behaviour around the production, detection and consumption of events and the responses they evoke. An EDA consists of:

- Creators - the source of the event.
- Consumers - entities that need to know an event has occurred.
- The Event Manager - an intermediary that, by applying rules, passes events to consumers.

The Event Manager decouples Creators and Consumers allowing them to broadcast and subscribe to different categories called channels. When an event occurs the Creator broadcasts the event on the Event Manager using a specific channel. A Consumer can subscribe to any number of channels on the Event Manager depending on events it needs to know about. When the Event Manager receives an event on a channel it sends the message to all the Consumers subscribed to that channel.

An EDA is especially suited to systems where the flow is determined by events that occur concurrently as opposed to a sequence of processes, where each process does not start until the one before it finishes.

A trading system lends itself to an EDA because:

- Changes in market data, orders and user commands can all occur concurrently - by expressing them as events the relevant part of the system can react with greater responsiveness.
- Each part of the system is decoupled from the rest of the system - this makes it easy to change each part without affecting the rest of it and to test each part independently by sending it events and making assertions on its behaviour.
- Each part of the system can be distributed over different locations and hardware - as the system changes each part of it can be scaled independently.

One of the disadvantages of an EDA is that the flexibility can lead to complexity as the system grows. A single event could trigger a number of Consumers making it hard to track the flow of the system. The design of the Core-Server tries to mitigate this issue by assigning distinct responsibilities to each part of the system.

Responsibilities of the Core-Server Components

Controller

All commands to the Core-Server are placed with the controller. The controller:

- Subscribes processes to the correct channels on the Event Manager when they request information.
- Un-Subscribes processes from channels on the Event Manager when they terminate or no longer require notification of specific updates.
- Routes commands to the correct parts of the Core-Server by broadcasting them on the correct channels on the Event Manager. [Appendix B](#) contains a list of all the commands supported by the controller.

Auto Trader

The Auto Trader:

- Runs automated trading strategies on selected markets.

- Broadcasts updates on the strategies state and progress.
- On request broadcasts a list of all running strategies.

Data Store

The Data Store:

- Records market data by saving it to the database.
- Replays stored market data by loading it from the database and broadcasting it.
- Records trading activity by saving it to the database.
- Replays stored trading activity by loading it from the database and broadcasting it.

Data Provider

The Data Provider:

- Polls the service layer for market data updates and broadcasts these updates to the Data Model.
- Ensures that all the markets currently subscribed to by other processes are being polled for updates.
- Ensures that the frequency of market data requests to the service layer do not exceed 5 per second per market.
- Groups the requests for market data so that the minimum number of requests are sent to the service layer but that the size of these requests does not exceed the exchange's market data limits[36].

Data Model

The Data Model

- Listens to market data updates from the Data Provider and stores the latest copy for each market.
- Every time data for a market is received that is different from the copy stored locally the Data Model broadcasts the new data. This ensures that subscribers to market data do not receive unnecessary updates when the state of a market hasn't changed.
- On request the Data Model will send a copy of a market's data to a requester. This ensures that new subscribers to a market don't have to wait till the next time the data changes to receive the market's data.

Order Manager

The Order Manager:

- Places and cancels orders with service layer.
- Subscribes to updates for all markets with unmatched orders and broadcasting events when these orders are updated or matched.
- On request broadcasts a list of all the user's unmatched orders.

Navigation Data Service

The Navigation Data Service:

- Downloads and stores the Betfair Application Navigation Data from the service layer.
- Periodically updates the Betfair Application Navigation Data by downloading a new copy from the service layer - this is essential as over time new markets are added and expire on the exchange.
- On request broadcasts the Betfair Application Navigation Data.

The Betfair Application Navigation Data is a data structure containing the hierarchy of all the events, markets and selections on the Betfair Exchange. It is required so the user can navigate and select which markets to trade.

8.1.4 The Service Layer

The Service Layer separates the Core-Server from the Betfair Exchange API and exposes a set of commands to the Core-Server. When the Core-Server actions one of these commands the Service Layer makes the required calls to the Betfair Exchange and returns any responses back to the Core-Server. By separating the Core-Server from the Betfair Exchange it decouples the Betfair Exchange's API from the Core-Server.

The advantages of this are:

- If the Betfair Exchange's API changes, as it did in 2014[8], only the Service Layer needs to be changed.
- Any future need for the trading system to communicate with another betting exchange could be achieved by only changing the Service Layer.
- Calls to the Betfair Exchange can be tested independently from the Core-Server.
- Modules in the Core-Server communicating with the Service Layer can be tested in isolation from the Betfair Exchange by replacing the Service Layer with a mock. Assertions can be made on the calls made to the Service Layer and the behaviour of the module based on the responses.

One of the requirements of the trading system is that it must be able to test a trading strategy without placing orders at the exchange. This is achieved by replacing the Service Layer with a Test Service Layer which manages all the orders placed by the Core-Server locally. As this provides the same set of commands to the Core Sever as the normal Service Layer to the Core Server the Core-Server's implementation does not need to be changed in order to conduct the test.

An explanation of how these orders are managed by the Test Service Layer is included in the implementation section.

8.2 Implementation

The Core-Server is written in Scala and built on Akka. Akka is a toolkit and runtime for building highly concurrent distributed and resilient message-driven applications[17]. Akka is based on Erlangs Actor model. An Actor provides a container for mutable state or behaviour. Other parts of

the system can communicate with an Actor by sending it messages which are queued and actioned one at a time. This has two key benefits:

1. It is not possible gain access to an Actor's mutable state from the outside unless the Actor publishes it so it is not necessary to synchronize access to parts of the code using locks. The developer can write code in an Actor without worrying about concurrency.
2. An Actor can communicate remotely with other Actors in different locations and in different applications so it is easy to distribute a system over resources.

Akka provides a toolkit for testing Akka components which includes mocks that can be used to mimic implementations of all the components and an environment where messages can be sent between Actors in sequence and assertions made on the nature and timing of the messages.

Each module in the Core Server is implemented as an Akka Actor. On initialisation the server completes the following tasks:

- Loads the Central Configuration file.
- Initialises the Service Layer.
- Authenticates to the Betfair Exchange and stores the session token.
- Initialises the Event Manager.
- Initialises the other modules passing them the central configuration and when required the session token.
- Subscribes each of the modules to the correct channels on the Event Manager.

8.2.1 Central Configuration

Every module on the server has an immutable configuration object injected into it. This object provides a central place to list the servers settings and an easy way to mock those settings. It includes:

- Names of the channels used on the Event Manager.
- The session key and application key for communication with the Betfair Exchange.
- The URLs for the Betfair Exchange.

8.2.2 Service Layer

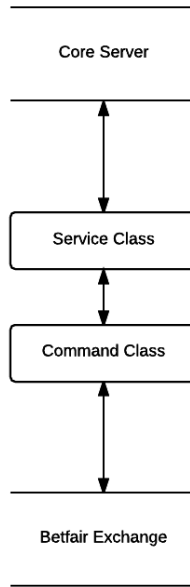


Figure 8.6: Service Layer Message Flow

The Service Layers message flow is shown in Figure 8.6. The Service Layer's implementation is split into three parts:

- Domain code - a package of Scala case classes, one for every type defined in the Betfair Exchange's API documentation[33].
- Service class - implements the interface exposed to the Core Server. When called this class creates a Json RPC request object using the Domain code and calls the Command class.
- Command class - posts http requests to the Betfair Exchange and converts the responses from Json to the correct Scala classes using the Domain code. All requests to the Betfair Exchange are sent as Json remote procedure call (RPC) requests and all responses received as Json RPC responses.

Test Service Layer

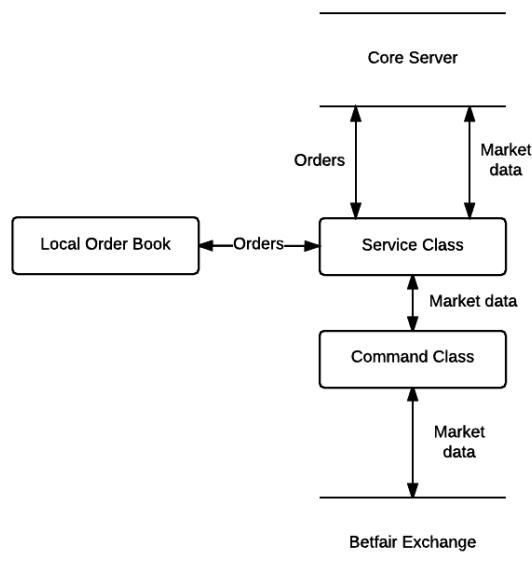


Figure 8.7: Test Service Layer Message Flow

A second version of the Service Layer called the Test Service Layer was implemented so that a trading strategy could be tested without placing orders at the Betfair Exchange. The message flow for the Test Service Layer is shown in Figure 8.7.

- All calls to the Test Service Layer to place, cancel, or change orders are sent to a locally maintained order book instead of to the exchange.
- Every time the Test Service Layer receives market data from the exchange it matches any orders that would have been matched based on the market data.
- All market data responses to the Core Server are amended to include the orders in the local order book.

This behaviour ensures that responses to the Core Server from the Test Service Layer are identical to those that would have been received if the orders were placed at the exchange. On start-up the Server decides which service layer to use based on a setting in the central configuration.

Local Order Book

When orders are placed at the exchange at a given price they are queued behind the orders placed previously. Orders are matched using a first in, first out (FIFO) method with orders at the front of the queue being filled first and those at the back last.

Because the local order book doesn't have access to the exchanges order book it cannot calculate the queue position of its orders. When the order was placed there may have been a volume of £500 before it but as this volume changes the local order book doesn't know if the volume was matched

or cancelled. So the local order book assumes its orders are at the back of the queue and only matches orders when the market data shows unmatched orders on the opposite side of the book. This provides a 'worse case' scenario when testing a trading strategy as local orders can take longer to get matched than those at the exchange.

The local order book maintains two queues for each market, one of Lay and one of Back orders. When the market data updates the local order book compares these queues to the market data, matches orders and updates the market data accordingly.

The Service Layer was implemented by extending the Betfair Scala example code[32] which demonstrates several of the Http calls to the exchange. The code uses Spray[21], a Scala toolkit for building REST/Http calls built on Akka, to send calls to the exchange. The conversions and validation from JSON to Scala case classes and visa-versa are implemented using Play's ScalaJson library[12].

The domain code was tested using ScalaCheck to assert that a Scala case class converted to JSON and then converted back to a Scala case class still has the same properties after the process as before. ScalaCheck allows the programmer to make asserts on functions using randomly generated data as opposed to conventional unit test where the data used is deterministic and outlined in the test.

The Command Class was tested by

- Mocking the central configuration, changing the exchange URL, so the request is sent to a socket on the local host, not the exchange.
- Asserting that the requests contained the session token and application key passed to them.

The Service Class was tested by

- Mocking the Command class
- Asserting that correct calls were made to the Command class based on the calls to the Service class.

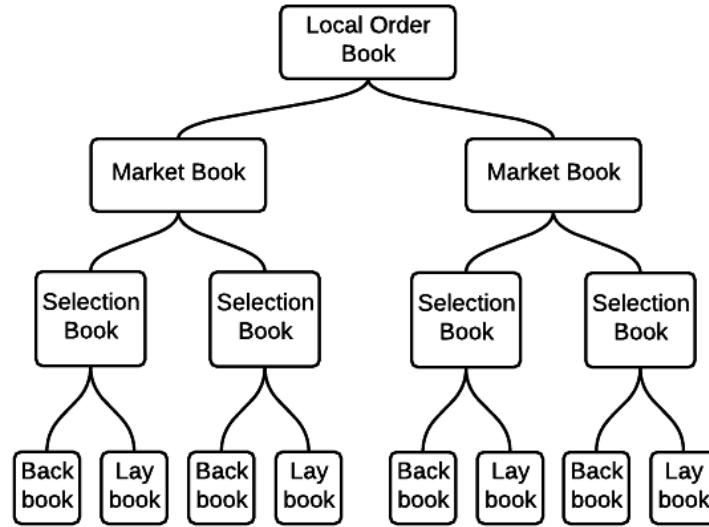


Figure 8.8: Local Order Book Hierarchy of Objects

As shown in Figure 8.8 the local order book is built from a hierarchy of objects.

- State tests were written for the Back/Lay book asserting that they persisted and matched orders correctly.
- Behaviour tests were written for all the objects above them, mocking the layer below and asserting that for each call the correct calls are made to the mocks below them.

8.2.3 Event Manager

The Event Manager was implemented by extending the Akka Event Bus. The main limitation of using the Akka Event Bus is that it can only communicate with other Akka Actors. Rabbit MQ[29] was initially considered as a solution but rejected as an unnecessary dependency because all the components that communicate with the Event Manager are implemented as Actors.

Extra functionality was added to the Akka Event Bus to support sub-channels with wild-cards allowing an Actor to subscribe to a subset of a channels data. For example:

- Market data for Market_B in Event_A are broadcast on the channel MarketUpdates/Event_A/Market_B
- If a subscriber only requires updates for Market_B they subscribe to MarketUpdates/Event_A/Market_B
- If the subscriber requires updates for Market_B but don't know the event Id they can subscribe to MarketUpdates/*/Market_B
- If the subscriber requires updates for all markets the subscribe to MarketUpdates/
- If the subscriber requires updates for all markets in Event_A they subscribe to MarketUpdates/Event_A/

This system reduces the amount of unwanted messages sent by the Event Manager.

The Event Manager was tested by subscribing a mock Actor to different channels, broadcasting events on a range of channels and asserting that the mock Actor only receive the correct events.

8.2.4 Controller

All requests to the Core Server are sent to the Controller. The controller broadcasts received commands on the correct channels on the Event Manager and subscribes Actors to the correct channels.

When the Controller receives a message to subscribe an Actor for market data updates it

- Subscribes the Actor to the required channel on the Event Manager.
- Broadcasts a message to the Data Provider informing it of the Actors subscription.

This behaviour ensures that the Data Provider knows who requires what market data and therefore knows which markets to poll for data.

When the Controller receives a command it starts watching the sender for termination. This is a facility provided by Akka where one Actor can request a message when another terminates. When the controller receives a termination messages it

- Un-subscribes the terminated Actor from all channels on the Event Manager.
- Broadcasts a message to the Data Provider, telling it to cancel their subscriptions.

and

The Controller was tested by mocking the Event Manager, sending the Controller messages and asserting that correct events were broadcast on the Event Manager.

8.2.5 Data Provider

The Data Provider is responsible for polling the service layer for market data. The Data Provider supports calls for two types of data:

- Market book data - dynamic data about markets, including:
 - Prices.
 - Status of the market.
 - Status of the selections.
 - Traded volume.
 - Current exchange prices for each selection in the market
 - Status of any of the systems orders for each selection in the market.

This call needs to be made at sub-second intervals to ensure the system has accurate market data.

- Market Catalogue data - a list of information about a published market that does not change, including
 - Name of the market.
 - Name of all the selections in the market.
 - Start time of the market.

This call only needs to be made once for each market.

Requests for market catalogue data are sent on demand when ever the Data Provider receives a command. Requests for market book data need to be throttled and grouped to ensure the exchanges limits aren't exceeded.

Market Book Data

The Betfair Exchange Documentation states:

- Calls should be made up to a maximum of 5 times per second to a single marketId[34].
- The number of calls be reduced by requesting data for multiple markets in one call - but there are limits on the amount of data that can be returned in one request.

PriceData	
Value	Description
SP_AVAILABLE	Amount available for the BSP auction.
SP_TRADED	Amount traded in the BSP auction.
EX_BEST_OFFERS	Only the best prices available for each runner, to requested price depth.
EX_ALL_OFFERS	EX_ALL_OFFERS trumps EX_BEST_OFFERS if both settings are present
EX_TRADED	Amount traded on the exchange.

Figure 8.9: Market Book Price Projections

listMarketBook

PriceProjection	Weight
Null (No PriceProjection set)	2
SP_AVAILABLE	3
SP_TRADED	7
EX_BEST_OFFERS	5
EX_ALL_OFFERS	17
EX_TRADED	17

Figure 8.10: Market Book Market Data Limits

The amount of data returned in a request is defined by the price projection, which is passed as a parameter to the exchange when a request is made. It stipulates the level of data to be returned for each market in that call. Figure 8.9 shows the price projections permitted by the exchange and Figure 8.10 shows the market data limits imposed by the exchange for each price projection. The Data Provider supports the combinations of price projection shown in Table 8.1

Price Projection	Markets/Call
EX_BEST_OFFERS	40
EX_ALL_OFFERS	11
EX_BEST_OFFERS & EX_TRADED	9
EX_ALL_OFFERS & EX_TRADED	5

Table 8.1: Number of markets/call by price projection

Price Projection	Amount of market data	Inclusive of
EX_BEST_OFFERS	3 prices deep	
EX_ALL_OFFERS	full price depth	EX_BEST_OFFERS
EX_BEST_OFFERS & EX_TRADED	3 prices deep and traded volume	EX_BEST_OFFERS
EX_ALL_OFFERS & EX_TRADED	full price depth and traded volume	EX_BEST_OFFERS EX_ALL_OFFERS EX_BEST_OFFERS & EX_TRADED

Table 8.2: Price projection hierarchy

Table 8.2 shows the amount of data returned the hierarchy of the combinations of price projections. To ensure the Data Provider does not exceed data limits or the number of calls per second and requests the correct price projection for each market it implements the following process:

- The Data Provider maintains a set of all the subscribers for each market and the price projections requested.
- At a configurable interval of no more than 250 milliseconds the Data Provider produces a list of all the required markets with the lowest weighted price projection necessary to accommodate each market's subscribers.
- This list is grouped by price projection and split into the maximum sized chunks permissible for each projection.
- The chunks are sent via an Akka Round Robin router[20] to a pool of Actors shown in Figure 8.11 (the number of Actors is set in the central configuration).
- These Actors send the requests concurrently to the Service Layer and when the data is received broadcast it on the corresponding channel on the Event Manager.

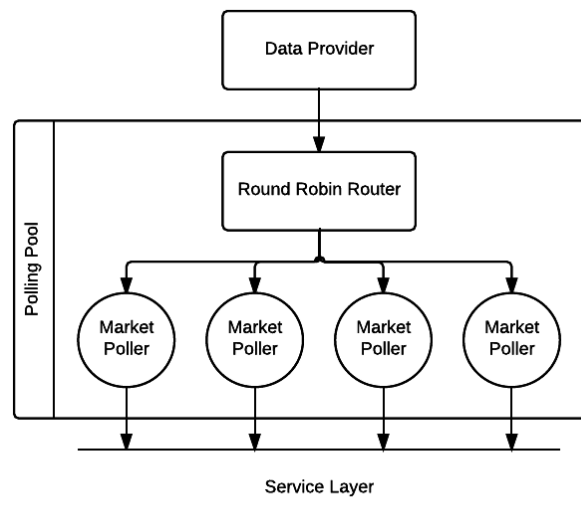


Figure 8.11: Data Provider Polling Mechanism

Each component in this process has been unit tested to ensure it correctly implements the process.

8.2.6 Data Model

The Data Model is implemented as an Actor which subscribes to the channel of market data on the Event Manager broadcast by the Data Provider and implements the behaviour described in the design section [8.1.3](#).

The Data Model in conjunction with the Data Provider ensures that an Actor can subscribe to market data updates and receive updates in real time when the data changes.

8.2.7 Web Server and Client

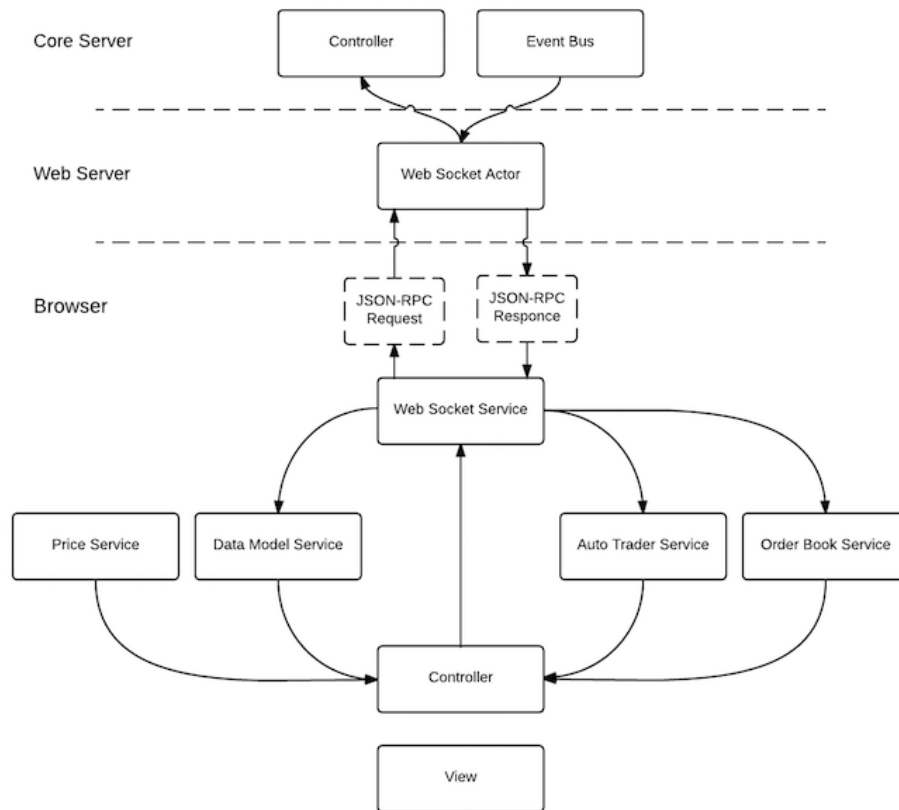


Figure 8.12: Web Server and Client Architecture

The Web Server and Client Architecture is shown in Figure [8.12](#).

- The Web Server communicates with the Core Servers Controller using Akka remoting[\[19\]](#).
- When the Client loads the web page from the Web Server it requests a web-socket for future communication.
- The Web Server creates an Actor to handle communication over the web-socket which it injects with a reference to the Core-Server's Controller. The web-socket Actor has two responsibilities:

1. Convert JSON-RPC requests sent from the the browser, over the web-socket, into Scala case classes and forward them to the Core Servers Controller.
 2. Convert Scala case classes sent from the Core Server into JSON-RPC responses and forward them over the web-socket to the browser.
- Because the web-socket is handled by an Actor it can be subscribed, by the Core Server's Controller, to the Event Manager.
 - When the web-socket Actor receives events from the Event Manager it converts them and pushes them over the web-socket to the Client.

All communication on the Client side is handled by an AngularJs service called the Web Socket Service which:

- Exposes an interface to other modules on the Client so that they can to send requests to the Web-Server.
- Distributes data received to the following AngularJs services:
 - Data Model Service - market price data.
 - Order Book Service - unmatched and matched orders.
 - Auto Trader Service - information about running strategies.

The price service contains logic for calculating Betfair price increments which are different depending on the current price (see Appendix C).

The Client was written in CoffeeScript (which compiles to JavaScript) using Google's AngularJs framework. Initially ScalaFx[13] and ScalaJs[24] were considered as alternatives but they were rejected as:

- The developer has written several medium sized web applications in AngularJs
- AngularJs is well documented, the others are not.
- AngularJs with a large on-line community, providing added support, which speed up the development process.

The Web Server was implemented using Play Framework[11] which was chosen because:

- It integrates with existing dependencies - it is built on Scala and Akka and the Server already had a dependency on Play's ScalaJson library, part of the framework.
- It compiles CoffeeScript into JavaScript in real time as it's being developed.
- It supports non-blocking IO over a RESTful API and web socket.

8.2.8 Navigation Data Service

The Navigation Data Service is responsible for downloading this file on start up, caching a copy of the file locally and updating that cached copy on an hourly basis.

The Navigation Data Service ensures the Core-Server has an up-to-date copy of the navigation data by implementing the following functionality:

- Downloads and caches a copy of the data on start up.
- Every hour downloads and caches a new copy the data.
- On request broadcasts navigation data for a given event type (i.e. Football or Horse Racing).

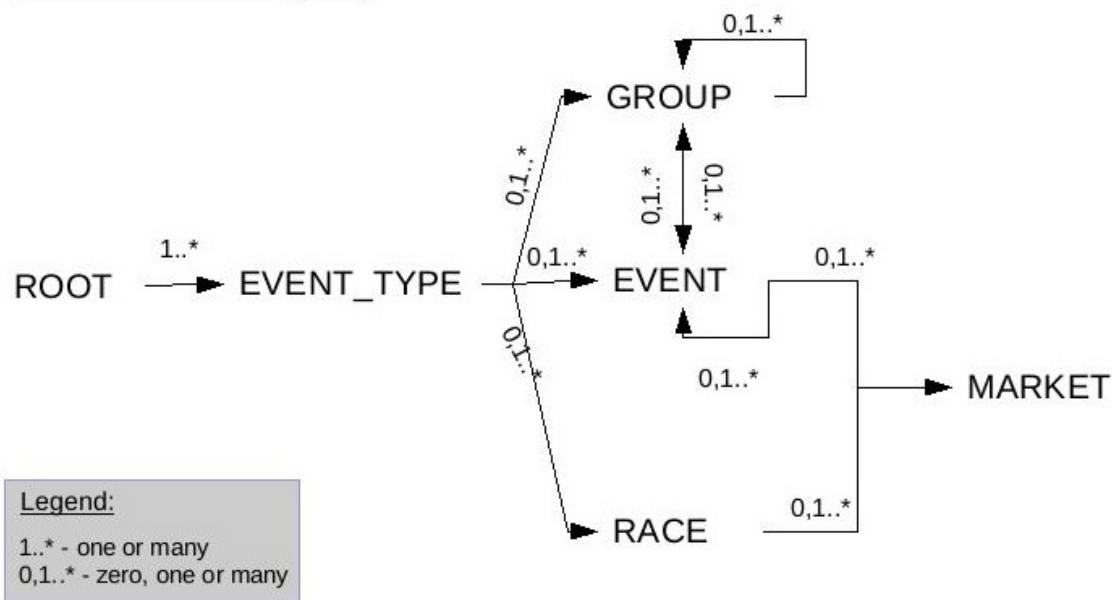


Figure 8.13: Navigation Data File Structure

The structure of the navigation data is shown in Figure 8.13. A package of Scala objects were written to mirror the structure of the file. When the file is downloaded it is validated and converted using Play’s ScalaJson library to a data structure of these objects that can be easily searched and filtered.

8.2.9 Order Manager

The Order Manager is responsible for:

- Making requests to place, cancel, update and replace orders with the Service Layer.
- Tracking and broadcasting updates on the status of orders once they have been placed with the Service Layer.
- Keeping a record of the volume of back and lay orders matched for each market and broadcasts updates when these matches occur.

The Order Manager is also responsible for and

On start up the Order Manager creates the initial list of tracked order by requesting a list of all the user’s unmatched orders at the Betfair exchange. At a configurable interval the Order Manager repeats this process reconciling its tracked orders with the list received from the exchange.

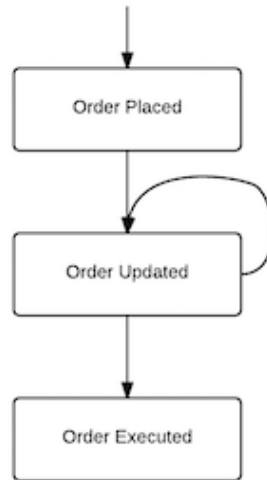


Figure 8.14: Life Cycle of an Order

Figure 8.14 shows the life cycle of an order.

- Once an order has been successfully placed with the Service Layer the Order Manager adds it to its list of tracked orders and tells the Controller to subscribe it to market data updates for the given market.
- As the Order Manager receives updates it compares the version of the orders in the update to those currently being tracked. If there is a difference the Order Manager will broadcast the new status of the order.
- Once the order has been executed the Order Manager will remove it from its list of tracked orders. If there are no more tracked orders for that market, it tells the Controller to unsubscribe it from further market updates for the given market.
- Market data updates include the total volume of matched back and lay orders for each selection and the average price they were matched at. The Order Manager keeps a list of these matches and when the matches in the market data update differ from those it holds the Order Manager broadcasts a message alerting subscribers to the update.

The Order Manager was tested by

- Mocking the Service Layer, sending requests to place and cancel orders to the Order Manager and making assertions on its calls to the Service Layer.
- Mocking the Event Manager, sending market data updates to the Order Manager and making assertions on the messages broadcast on the Event Manager.

8.2.10 Auto Trader

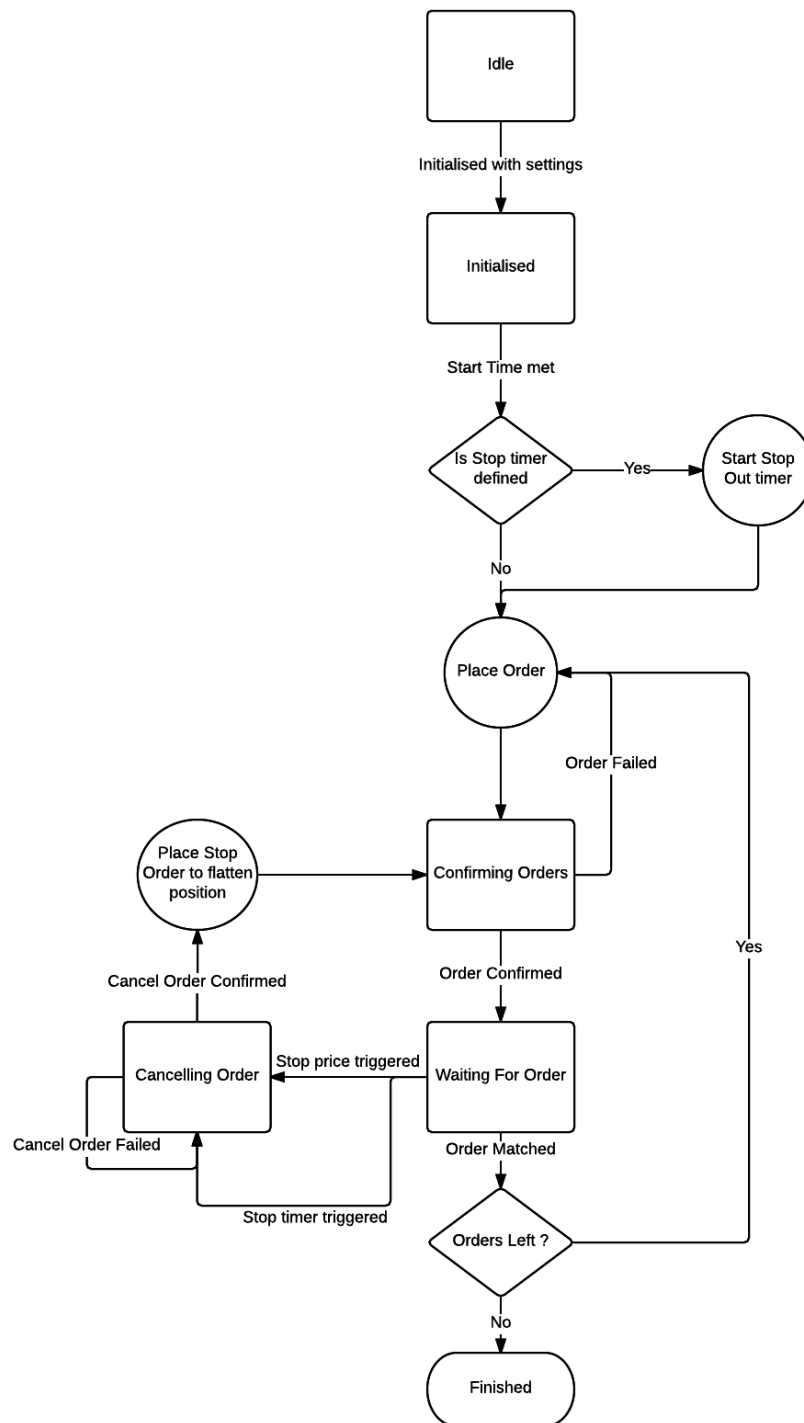


Figure 8.15: Basic Trading Strategy

Figure 8.15 shows a flow chart of a basic trading strategy that places a list of predefined orders and stop outs after a given time or if a given price is reached. The strategy can be defined as a series of states with events triggering actions and changes in the state. In Figure 8.15 States are shown in the rectangles, actions in the circles and the events as the lines connecting them:

$$State(w) + Event(x) = Action(y) + State(z)$$

Reducing the Strategy to a set of states, events and actions makes it:

- Simple to describe.
- Easier to test - because each state can be tested by applying events and testing the resulting actions and state.
- Easier to integrate into the trading system's Event Driven Architecture.

The Strategy

The strategy was implemented using an Akka FSM (Finit State Machine)[18] which is a mixin for the Akka Actor. An Akka FSM allows the developer to define a set of states, which state to start in and for each state define a set of events and the actions and state transitions they trigger. The strategy is given access to the Core-Servers Controller and sends it commands and receives updates from the Event Manager in the same way the client does. An instance of the strategy is created every time the user instructs the server to run the strategy on a new market. On initialisation the strategy is sent instructions that include the:

- Start time for the strategy to commence trading.
- Orders to be placed in the to be actioned.
- Prices to place them.
- Price at which to stop the strategy out.
- End time at which the strategy must stop out.

The Monitor

The monitor watches the strategy and broadcasts messages on the Event Manager when the strategy changes state. A new instance of the monitor is created for every strategy. Separating the reporting of a strategy's state from its implementation reduces the complexity of the strategy making it easier to build and test.

The Auto Trader

The Auto Trader listens on the Event Manager for commands from the Controller to:

- Run the strategy on a new market with a given set of instructions.
- Stop running a strategy on a market.
- Broadcast a list of all the strategies currently running.

The trading strategy was tested by:

- Mocking the Controller.
- Setting the strategy's state and data.
- Sending it events and making assertions on the commands sent to the controller and the strategy's resulting state and internal data.
- Each combination of State and Event is tested ensuring the strategy exhibits the correct behaviour in all states.

8.2.11 Data Store

Although the Data Store is included in the Design section of this document it is still under construction. The decision was made to use MongoDB as the database for the data store and to create the predictive model because it can store different objects in JSON format without having to define a schema for each type of object.

Predicting Price Movement

This part of the project will apply technical analysis indicators to historical Betfair price data. It will train an extremely random forest to predict the future direction of horse racing markets based on their previous price movement. The following sections will:

- Describe the data and how it was manipulated in order to apply the indicators.
- Provide an explanation of the indicators used and the reasons for using them.
- Describe the algorithm used to train the model and the reasons for selecting it.
- Describe how the model was implemented and optimised.
- Show and discuss the results of testing the model.

9.1 The Data

The data is comprised of horse racing markets from the 11th October 2014 to the 10th November 2014. The data was produced by polling the Betfair exchange in 250 millisecond intervals with a new record created for each selection every time there was a delta in the price data. The data consists of 97,422,530 records over 847 markets (races) or 8358 selections (runners). The data was supplied as a csv file (see [Appendix D](#) for schema).

The data was read into a MongoDB database, splitting the data into collections, one for each market.

The data was aggregated by market into a series of 1 minute intervals, recording the high, low and closing price, the volume traded over the interval and the weight of market at the end of the interval. The weight of market is:

$$\frac{\text{sum of the volume at the 3 highest prices available to back}}{\text{sum of the volume at the 3 lowest prices available to lay}}$$

Betfair prices are represented as decimal odds unlike financial markets where prices are quoted in pence (for securities traded in sterling). As [Table 9.1](#) shows a move from 1.01 to 2.00 has the same effect on the probability as a move from 2.00 to 100.0. Betfair addresses this by changing the size of the increments between prices (referred to as ticks) as the prices increase. [Appendix C](#) contains a table of Betfair price increments. The technical analysis indicators used in this project were designed to be used on financial markets where the size of the ticks is generally constant. For this reason and to stop the model being biased towards movements above 2.00, all prices were converted from decimal odds to implied probabilities.

Decimal Odds	Implied Probability	
	For	Against
1.01	99.0	1.0
1.50	66.6	33.3
2.00	50.0	50.0
3.00	33.3	66.6
100.00	1.0	99.0

Table 9.1: Implied Probability of Decimal Odds

9.2 The Indicators

The indicators used in this project are widely used in financial trading. Appendix E contains a list of all the technical analysis indicators used in this project with a web link to the Stock Charts website for a more detailed description of their history, how they are calculated and interpreted.

The indicators were selected because their outputs all oscillate within a range and as a result can be compared across markets. A major difference between financial markets and horse racing markets is that a horse racing market has a much shorter life than a financial market. Models trained on financial markets are usually trained with data from the same market they predict. This is not possible with the horse racing markets due to their short life span as there isn't enough data. It is important that the indicators can be compared across markets because the model in this project will be trained using data from many markets.

Each indicator was implemented in Scala and tested by creating unit tests with example data downloading from the Chart School website[39] and asserting that given the input in the example data they produced the same output.

On its own each of these indicators is unlikely to accurately predict a price trend in all markets, all the time, but it is hoped by using a combination of indicators the model will better predict the trend.

9.3 The Algorithm

This project uses a Random Forest to classify the data. A Random Forest was selected because:

- They have shown excellent performance for both classification and regression problems.
- They are expressive and easy to understand[10].
- They are easy to implement due to their recursive nature[10].
- They can handle classification problems with more than two output classes.

A Random Forests consists of an ensemble of decision trees each built from samples of the training data drawn with replacement and grown using random subsets of features. Random Forests were initially proposed in the 2001 paper Random Forests by Leo Breiman[5].

9.4 Decision Trees

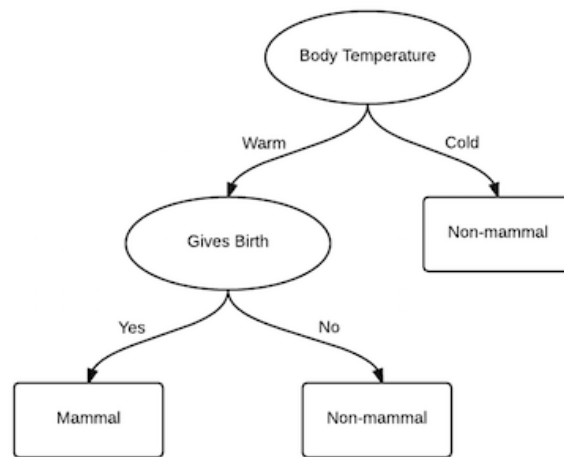


Figure 9.1: Decision Tree for Mammal Classification

Decision Trees classify data by learning simple decision rules inferred from the data. Decision Trees are grown by splitting the data, one feature at a time, selecting the feature that produces the best split of the data. This procedure is repeated until the data is of a single classification or a set size. The quality of the split is commonly decided by maximising the entropy gain but other measures can be used. Figure 9.1 shows an example of a simple decision tree for classifying mammals.

Advantages of decision trees:

- Simple to understand.
- Able to handle both numerical and categorical data.
- Able to handle problems with more than two output classes.
- Easy to interpret as the explanation can be explained using Boolean logic (in contrast to a neural network)

Disadvantages of decision trees:

- Can be prone to over-fitting the training data as they do not generalise well over the data and as a result exhibit low bias. For example: All the mammals in your training data may have green eyes whilst the non-mammals have blue eyes. The decision tree would conclude that all mammals have green eyes.
- Can exhibit high variance as small changes in the data can result in a completely different model.

9.5 Random Forest

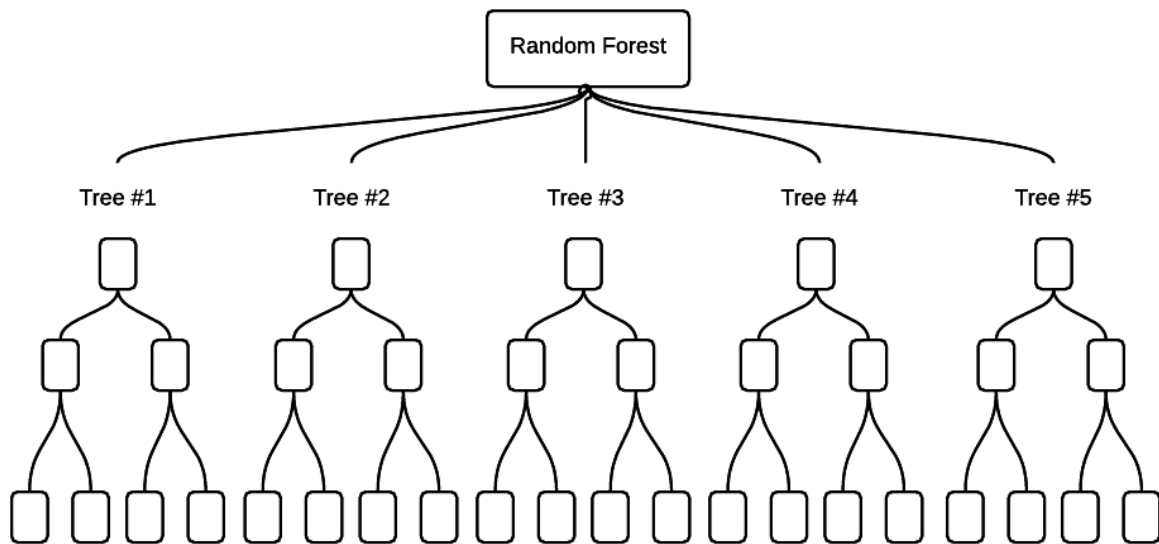


Figure 9.2: A Random Forest with 5 trees

A Random Forest is an ensemble of decision trees. It classifies data by taking the most popular class after each tree in the forest has classified the data independently. For example: if the forest in Figure 9.2 was used for the classify a mammal 3 of the trees might predict Mammal and the 2 of them Non-mammal. The random forest would select mammal as it was the majority class voted for by its trees.

- Each tree in a Random Forest is grown using a subset of the training data, the same size as the training data, but selected at random for the initial dataset using replacement. These subsets are known as bootstrap samples.
- As each tree is grown the algorithm only uses a random subset of features at each split to divide the data. This is known as subspace sampling.
- Trees are fully grown, without pruning until:
 - All the instances in a node are of the same class or,
 - All the features in a node are of the same value or,
 - The number of instances in a node is less that the minimum number to split on

A Random Forest has three configurable options that can be optimised during the training process:

- The number of trees in the forest.
- The number of random features to choose from at each split.
- The minimum number of instances to split on.

Introducing randomness into the trees through the bootstrap samples and subspace sampling encourages diversity within the forest. Due to averaging, the trees that are stronger predictors in the forest have a greater effect on the overall outcome with the decisions from the weaker trees cancelling each other out. As a result the bias of the forest is usually slightly higher than that of a single non-random decision tree but the variance also decreases which, in most cases more than compensates for the increase in bias.

9.5.1 Overhead of Training a Random Forest on Features with Continuous Values

Features		Class
A	B	
Red	1	Mammal
Red	2	Mammal
Red	3	Mammal
Red	4	Non-Mammal
Red	5	Non-Mammal
Green	6	Non-Mammal
Green	7	Mammal
Yellow	8	Mammal
Yellow	9	Mammal
Yellow	10	Non-Mammal

Table 9.2: Labelled Example Data

When each tree is grown in a Random Forest the algorithm needs to select the optimal threshold to split each feature on, by comparing the entropy of each split, before it decides which feature to use to divide the data. If you consider the example data Table 9.2:

- Feature A only has three finite values that can be used to split the data, Red, Green or Yellow.
- Feature B, which contains continuous data, has 9 possible splits available.

As the size of the training data increases so would the number of possible values that feature B could be split on. For a dataset of n instances $n - 1$ splits would need to be considered for each feature (assuming all the values were unique). If three randomly picked features with continuous data were considered this would result in $3 * (n - 1)$ comparisons for each split. The data used for training in this project has 14 features all containing continuous data and over 100,000 instances in the training set. In order to speed up the training process a variation on the Random Forest was selected.

9.6 Extremely Random Forest

Extremely Random Forests[14] build on the Random Forests algorithm by introducing an extra level of randomness.

- As each tree is grown the thresholds to split each feature on are picked at random from the features range in the dataset, e.g. for Feature B in Table 9.2 a random number would be picked between 1 and 10.
- If 3 features were being compared at each split the entropy would be compared using a random threshold for each feature and the one that produces the best split chosen.

Selecting a value at random is computationally faster than comparing all the possible splits and therefore scales better. A random number picked in constant time will not be effected by the size of the dataset.

Figure 9.3 shows the algorithm to grow the trees used in the Extremely Random Forest.

Split_a_node(S)

Input: the local learning subset S corresponding to the node we want to split

Output: a split $[a < a_c]$ or nothing

- If **Stop_split(S)** is TRUE then return nothing.
- Otherwise select K attributes $\{a_1, \dots, a_K\}$ among all non constant (in S) candidate attributes;
- Draw K splits $\{s_1, \dots, s_K\}$, where $s_i = \text{Pick_a_random_split}(S, a_i), \forall i = 1, \dots, K$;
- Return a split s_* such that $\text{Score}(s_*, S) = \max_{i=1, \dots, K} \text{Score}(s_i, S)$.

Pick_a_random_split(S, a)

Inputs: a subset S and an attribute a

Output: a split

- Let a_{\max}^S and a_{\min}^S denote the maximal and minimal value of a in S ;
- Draw a random cut-point a_c uniformly in $[a_{\min}^S, a_{\max}^S]$;
- Return the split $[a < a_c]$.

Stop_split(S)

Input: a subset S

Output: a boolean

- If $|S| < n_{\min}$, then return TRUE;
- If all attributes are constant in S , then return TRUE;
- If the output is constant in S , then return TRUE;
- Otherwise, return FALSE.

Figure 9.3: Algorithm for growing a Decision Tree for an Extremely Random Forest[14]

9.7 Implementation

The following implementation options were considered but rejected:

- Using Apache Sparks MLlib[41]. This was rejected because it can only be used with Apache Spark and jobs must be submitted to a Spark cluster. Introducing Spark to the list of technologies used would reduce development speed.
- Using Python's SkLearn library[22] and calling it from Scala. This was rejected because of the overhead of communicating between Scala and Python[7].

The decision was made to implement the Extremely Random Forest from scratch in Scala.

9.7.1 Labelling the Data

Once the data had been grouped into intervals, the prices converted and the technical indicators applied to each interval, each interval was labelled as one of three classes: "Up", "Down" or "None" indicating the delta between the interval's close and the close of the next interval in the series.

The data was split into two sets; training and testing

- The training set comprised of data from markets between the 11th October 2014 and the 31st October 2014.
- The testing set comprised of data from the 1st November 2014 to the 10th November 2014.

9.7.2 Reducing the Size of the Data

The majority of the liquidity on horse racing markets on the Betfair exchange is introduced during the last 20 minutes before the start of a race. Without sufficient liquidity prices can move irrationally. Consequently the decision was made to only include intervals from the last 20 minutes of trading for each race in both the training and testing sets and to only use the model to predict price movements in the last 20 minutes of trading on any market. However the indicators were applied on the intervals *before* this was done as the majority of them are averaged over more than 20 data points and as such would require more than 20 minutes of data to output useful values.

9.7.3 Adjustments for Unevenly Distributed Data

Class	Number of Instances	
	Training Set	Testing Set
Up	36780	14918
Down	35117	13571
None	42198	14907
	114095	43396

Table 9.3: Training and Testing Datasets

Figure 9.3 shows the distribution by class of each dataset. The training data is not evenly distributed, even if it was there would be no guarantee that future training sets would be. Unevenly distributed training data causes the resulting model to favour the most frequently occurring classes.

There are two approaches to tackle this problem:

1. Assign a cost to misclassification, with a higher cost associated with the minority class whilst trying to minimize the overall cost.
2. Either down-sample the majority class or over-sample the minority class when creating each bootstrapped dataset for each tree[6].

This implementation uses the second of these approaches. The original dataset is divided into sets by class and each of the resulting sets sampled with replacement so the resulting bootstrapped set was the same size as the original dataset and has an equal distribution of class. This method could

result in majority classes being over-sampled and minority ones being down-sampled depending on the original dataset.

The trees are fully grown using the bootstrapped datasets until one of the following conditions are met:

- The number of instances is less minimum number to split on
- All the labels are of the same classification
- All the attributes are constant

The minimum number of instances to split on was set to 2 as is advised for classification problems, to reduce the chance of a leaf containing instances of the same class. If a leaf (end node of the tree) does contain instances of more than one class, which could occur if all the attributes were constant, the implementation selects one of those classes at random.

Once the forest has been trained it is serialised to disk for later use.

9.8 Optimisation

The number of features to split on and the number of trees in the forest were optimised to reduce to out-of-bag error over the training data.

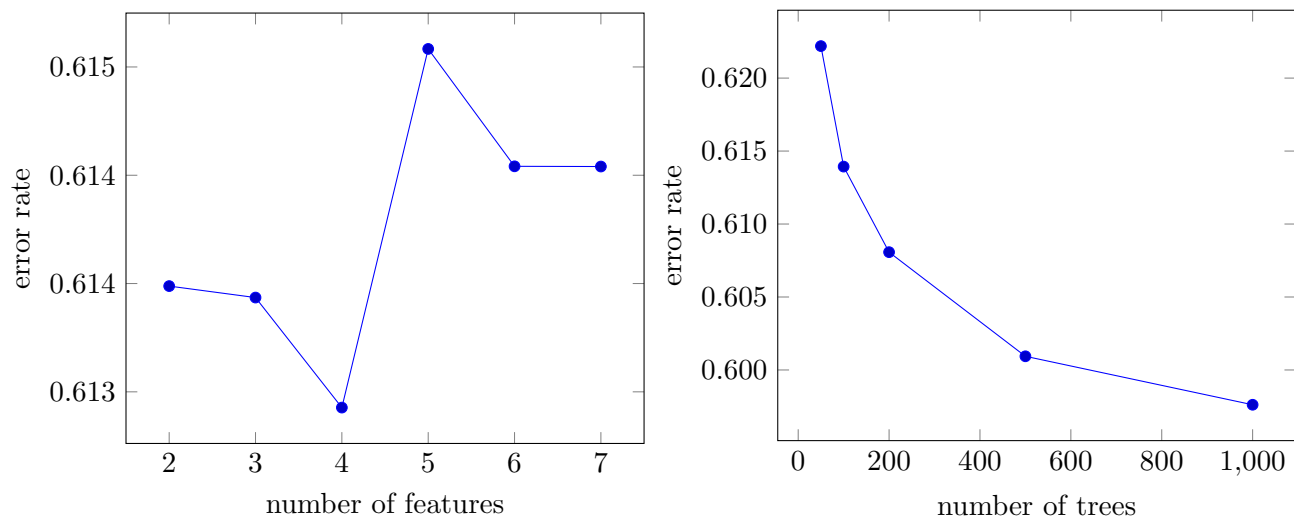
9.8.1 Out-Of-Bag Error

Each tree in the forest is grown using a bootstrapped sample. When this sample is created about 37% of the original dataset is left-out, this is known as the out-of-bag sample. Once the tree has been grown it is used to classify each instance in the out-of-bag sample. After all the trees have been grown these votes are aggregated and used to classify all the instances in the original dataset. The out-of-bag error rates is:

$$\frac{\text{number of mis-classified instances}}{\text{total number of instances}}$$

The out-of-bag error has been shown to be as accurate as using a test set the same size as the training set[4].

9.8.2 Optimisation Results



(a) error rate for features

(b) error rate for number of trees

Figure 9.4: Graphs of Optimisation Results

Features	Error Rate
2	0.613988
3	0.613935
4	0.613427
5	0.615083
6	0.614541
7	0.614540

(a) error rate for features

Trees	Error Rate
50	0.622192
100	0.613935
200	0.608072
500	0.600937
1000	0.597616

(b) error rate for number of trees

Table 9.4: Tables of Optimisation Results

The results were recorded using the default settings below, changing either the number of features or the number of trees.

number of trees 100
 minimum leaf size to split 2
 number of features to split on 3

As shown in Table 9.4a and Figure 9.4a the optimal setting for the number of features to split on to minimising the error rate was 4, which coincides with that recommended by Brieman, the square root of the total number of features.

As shown in Table 9.4b and Figure 9.4b the error rate decreases as the number of trees in the forest increases.

9.9 Results

		Predicted			
		Up	Down	None	
Actual	Up	6674	5254	2990	14918
	Down	4811	6377	2383	13571
	None	5106	4895	4906	14907
		16591	16526	10279	

Table 9.5: Predicted Classes Vs Actual Classes

Class	Distribution	Precision	Recall
Up	0.343764	0.402266	0.447379
Down	0.312725	0.385877	0.469899
None	0.343511	0.477284	0.329107

Table 9.6: Distribution, Precision and Recall by Class

Out-of-bag error	0.597265
Accuracy	0.413794
Weighted Average Precision	0.422910
Weighted Average Recall	0.413794

Table 9.7: Weighted Accuracy, Precision and Recall

The results in Figures 9.5 9.6 and 9.7 were produced training the classifier on the training set with the following settings selected during the optimisation stage and testing the resulting model on the testing set:

number of trees	1000
minimum leaf size to split	2
number of features to split on	3

The results show that the classifier exhibits better accuracy, precision and recall by class than could be achieved by selecting a class at random given the distribution. Precision is highest for the "None" class and lowest for the "Down" class. Recall is highest for the "Down" class and lowest for the "None" class.

Conclusion

The first part of the project has met the primary objectives to build a trading system that can be used to:

- Navigate markets on the Betfair Exchange.
- Manually trade markets on the Betfair Exchange.
- Automate the trading of markets on the Betfair Exchange.

The the first part of the project has not met the secondary objectives to add functionality to:

- Record market data and trading activity to be analysed after the fact.
- Provide a framework to back-test an automated trading strategy against recorded market data.

The second part of the project has met the primary objectives to:

- Select a machine learning algorithm to be used to build a model from historical market data.
- Process the historical market data into intervals and select and apply technical analysis indicators.
- Divide the data into sets that can be used to train and test the model.
- Optimise the model using the training data.
- Analyse the results of the model on the testing data.

The second part of the project has not met the secondary objectives to:

- Integrate the model into a trading strategy.
- Automate the trading strategy using the system built in the first part of this project.

Appendix [F](#) contains a list of issues raised during the development process to be implemented as future work.

Additionally the predictive model needs to be integrated into the trading system to test its accuracy on a live market. At the moment the results only show its accuracy on predicting individual intervals. A true test of the model would be to incorporate it into a strategy that included:

- Thresholds for stopping out losing trades.
- Thresholds for how much profit to take on winning trades.
- How many successive Up or Down indicators are required for a trade to be actioned.

The accuracy of the predictive model could be improved by:

- Changing the technical analysis indicators used as features.
- Experimenting with different combinations features and changing the number of features used.
- Using regression as opposed to classification, outputting real numbers showing the predicted probability that a market will trend up, down or stay the same.
- Comparing the results of other machine learning algorithms such as a Support Vector Machine or Multi layer Perceptron to see if they better suit the data.

10.1 Development Approach

Agile

The Agile approach to development has worked well. Implementing requirements in small iterative cycles ensured consistent progress during development.

Testing

The focus on continuous testing ensured that each commit didn't break the build. Using Git and SBT to track and manage files and builds was very effective. Due to time constraints there are areas of the code that still need testing. This was a result of trying to balance the speed of development with the level of testing to ensure the project's primary objectives were met. Issues have been raised for each section and are in [Appendix F](#).

Appendices

Glossary of Terms

A.1 Gambling Terms

Back

The term Back is used on betting exchanges and means that you bet ON a result happening. Placing a Back bet is the same as placing a bet with a normal book maker.

Lay

The term Lay is used on betting exchanges and means that you bet AGAINST a result happening. When placing a Lay bet you are taking the liability of the bet, in effect you are taking the role of a normal book maker.

Side of a Bet

Either Back or Lay.

Offering Odds

Placing a lay bet at a given price.

Stake/Size

The amount the bet is for.

Odds/Price

The amount returned, in respect to the stake, if a result happens. All prices on the Betfair Exchange and in this dissertation are displayed as decimal odds.

Decimal odds vs Fractional odds

Traditionally bookmakers in the United Kingdom have displayed prices as fractional odds, for example:

If the fractional odds are 3/1 and you place a back bet of £10 and you win, your total return would be:

$$£10 \times 3 \text{ as profit} + \text{your original } £10 \text{ stake} = £40$$

If the decimal odds are 4.0 and you place a back bet of £10 and you win, your total return would be:

$$£10 \times 4.0 = £40$$

Decimal odds can be easier to calculate than fractional odds with a denominator that is not 1, for example:

The fractional odds 6/5 are the equivalent of the decimal odds 2.2

The total return if you place a back bet of £10 at odds of 2.2 and you win would be:

$$£10 \times 2.2 = £22$$

This is easier than calculating:

$$\left(£10 \times \frac{6}{5}\right) + £10 = £22$$

The important thing to remember is decimal odds always include the unit stake thus every price will be greater than 1. Fractional odds represent the profit, 6/5 means you will win six pounds for every five pounds staked.

Profit

The amount you could profit

- For a Back bet this is calculated as $(price \times size) - 1$
- For a Lay bet this is the size of the bet

Liability

The amount you could lose

- For a Back bet this is the size of the bet you have placed.
- For a Lay bet it is the amount it will cost you if the selection wins, this is calculated as $(price \times size) - 1$

Return

Profit - Liability

Matched Bets

Bets that have not yet traded as they have not been paired with a bet of the same price from the opposite side.

Unmatched Bets

Bets that have been paired with a bet of the same price from the opposite side.

Orders

Bets

A.2 Machine Learning Terms

Entropy

A way to measure (im)purity.

$$Entropy = \sum_x -p_x \log_2 p_x$$

Where p is the probability of the class.

For Example:

Class
Red
Red
Red
Red
Red
Green
Green
Yellow
Yellow
Yellow

The probability of each colour in the set above is:

$$\begin{aligned}\text{Probability}(\text{Red}) &= \frac{5}{10} = 0.5 \\ \text{Probability}(\text{Green}) &= \frac{2}{10} = 0.2 \\ \text{Probability}(\text{Yellow}) &= \frac{3}{10} = 0.3\end{aligned}$$

So the entropy of the set is:

$$Entropy = -0.5 \log_2(0.5) - 0.2 \log_2(0.2) - 0.3 \log_2(0.3) = 1.485$$

Entropy reaches its maximum when all the classes have equal probability. The entropy of a set with only one class is zero.

Overfitting

Overfitting occurs when a model is too closely fit to the training data and does not generalise over the data. The line on right graph in Figure [A.1](#) over fits the data. This causes a problem because the training data is only a sample of the problem space and will contain noise. If a model overfits the training data it will have a low error when used to predict the training data but a high error when used to predict data outside of the training data.

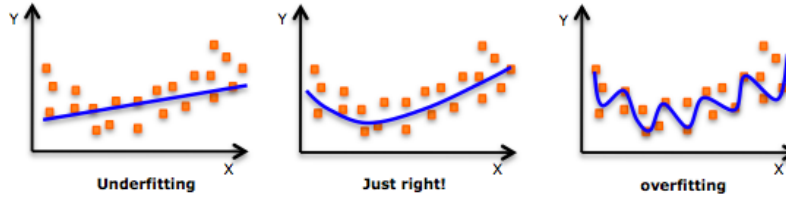


Figure A.1: Example of underfitting, rightfitting and overfitting[9]

Bias

Represents the ability of a model to approximate the data. A model that underfits would display high bias, a model that overfits would display low bias.

Variance

The sensitivity of a model to changes in the data points. If a model is retrained the variance is how much the model changes when the data points change.

Sampling with Replacement Vs Sampling without Replacement

Consider the set of numbers $[1, 2, 3]$

- Sampling without replacement - if two numbers are selected from the set without replacement the possible combinations are: $[1, 2]$, $[1, 3]$, $[2, 3]$, $[2, 1]$, $[3, 1]$, $[3, 2]$ because the first number is removed from the set before selecting the second number.
- Sampling with replacement - if you select two numbers from the set with replacement the possible combinations are: $[1, 1]$, $[1, 2]$, $[1, 3]$, $[2, 1]$, $[2, 2]$, $[2, 3]$, $[3, 1]$, $[3, 2]$, $[3, 3]$ because the first number is replaced in the set making it available for the second selection.

In sampling with replacement the sample values are independent, when sampling without replacement they are not.

Pruning a Decision Tree

Pruning a decision tree removes parts of a decision tree that have less effect on classification, reducing its complexity. Pruning is used to increase a trees accuracy by reducing overfitting.

Controller API

Command	Parameters	Description	Routed To
GetNavigationData	eventType	Request the navigation data for the given event type (e.g. soccer)	Navigation Data Service
PlaceOrders	marketId; instructions; reference	Place the list of orders defined in the instructions with the exchange on the specified market	Order Manager
CancelOrders	marketId; instructions; reference	Cancel the list of orders defined in the instructions with the exchange on the specified market	Order Manager
ReplaceOrders	marketId; instructions; reference	Replace the list of order defined in the instructions with the exchange on the specified market	Order Manager
UpdateOrders	marketId; instructions; reference	Update the list of orders defined in the instructions with the exchange on the specified market	Order Manager
ListMarketCatalogue	marketIds	Request the latest copy of the given list of market's catalogue	Data Provider
ListMarketBook	marketIds	Request the latest copy of the given list of market's data	Data Model
ListCurrentOrders	betIds; marketIds	Request a list of all unmatched orders with the given betIds on the given markets.	Order Manager
ListMatches		Request a list of all matched orders	Order Manager
SubscribeToSystemAlerts		Subscribe to system alerts	Event Manager
SubscribeToMarkets	marketIds; level of data	Subscribe to market data updates for the given markets for the given level of data	Event Manager and Data Provider
UnSubscribeFromMarkets	marketIds; level of data	Un-Subscribe from market data updates for the given market for the given level of data	Event Manager and Data Provider
UnSubscribe		Un-Subscribe from all channels	Event Manager and Data Provider
SubscribeToOrderUpdates	marketId; selectionId	Subscribe to order updates for the given market and selection	Event Manager
SubscribeToAutoTraderUpdates	strategyId	Subscribe to auto trader updates for the given strategy	Event Manager
StartStrategy	marketId; selectionId; config	Start running the strategy defined in the config on the given market and selection	Auto Trader
StopStrategy	marketId; selectionId	Stop running the strategy on the given market and selection	Auto Trader
ListRunningStrategies		Request a list of all running strategies	Auto Trader

Betfair Price Increments

Price	Increment
1.01 → 2	0.01
2 → 3	0.02
3 → 4	0.05
4 → 6	0.1
6 → 10	0.2
10 → 20	0.5
20 → 30	1
30 → 50	2
50 → 100	5
100 → 1000	10

Historical Data Schema

Field	Type
Event ID	String
Name of market	String
Market ID	String
Date	Date
Course	String
Race time	Date
Time stamp	Date
Inplay flag	Boolean
Market status	String
selection id	String
selection	String
total matched	Double
last price matched	Double
back price1	Double
back price2	Double
back price3	Double
back vol1	Double
back vol2	Double
back vol3	Double
lay price1	Double
lay price2	Double
lay price3	Double
lay vol1	Double
lay vol2	Double
lay vol3	Double

Technical Analysis Indicators

Name	Link
Accumulation Distribution Line	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:accumulation_distribution_line
Chaikin Oscillator	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:chaikin_oscillator
Commodity Channel Index	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:commodity_channel_index_cci
MACD	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:macd-histogram
Money Flow Index	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:money_flow_index_mfi
Rate Of Change	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:rate_of_change_roc_and_momentum
Relative Strength Index (RSI)	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:relative_strength_index_rsi
Stochastic Oscillator Percent K	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:stochastic_oscillator_fast_slow_and_full
Stochastic Oscillator Percent D	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:stochastic_oscillator_fast_slow_and_full
Stochastic Oscillator Slow Percent D	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:stochastic_oscillator_fast_slow_and_full
Williams Percent R	http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators:williams_r

Future Work

Area	Importance	Description
Core System	2	Implement System channel for alerts
Core System	3	Review exceptions in server
Core System	3	Remove marketbookupdate and add timestamp to marketbook
Core System	2	Move config into a file in resources
Core System	2	Review all TODOs in the code
Core System	2	Address all compilation warnings
Core System	2	Remove all print statements replace where necessary with log messages
GUI	3	Combine all css files convert to sass or less
GUI	2	Pagination for detail view this will reduce the volume of data being sent to/received from server
GUI	2	Implement Pub/Sub on Websocket service in angular
GUI	3	Add UI MarketCatalogue to webserver
GUI	2	Navigate data store
Autotrader	2	Action strategy on stored data
Datastore	2	Pick markets to store to db
Datastore	2	Review stored data
Testing	3	Add tests to build
Testing	2	Unit test Dataprovider
Testing	2	Finish Unit tests for order Manager
Testing	2	Unit Test webserver
Testing	3	Unit test GUI using Karma
Testing	2	Data utils unit tests need fixing
Order Manager	2	Order manager should broadcast warnings when limits are close
Order Manager	2	Order manager needs to be able to place orders below 2 in live markets
Order Manager	3	Order manager should be split into order manager and order tracker
Order Manager	2	Add executed orders to order manager so it can track overall pnl
Data Utils	3	Create bespoke execution context for reactive mongo to use
Data Utils	3	Profile Random Forest
Sim Order Book	2	Refactor Sim Order Book to simplify
Sim Order Book	2	Rewrite match orders logic to simplify
Sim Order Book	1	Order Book should clear down orders when the market closes
Data Provider	1	Fix market polling errors for closed markets
GUI	1	Navigate horse racing
Data Utils	1	Save classifier and test on live market
GUI	1	Cashout and pnl is wrong on ladder
GUI	1	Increase ladder size as window size changes
Auto trader	1	Check if market is suspended when making autotrader decisions
Auto trader	1	Write scalping strategy using machine learning model
GUI	1	GUI Page for running strategies
Sim Order Book	1	Sim Order Book should pull orders when market is suspended
Sim Order Book	1	Sim Order Book should cancel orders for closed markets
Auto trader	1	Autotrader handle orders being cancelled manually by the user
Auto trader	1	Autotrader stop when market is closed

Bibliography

- [1] Betfair. *A Betting Revolution*. 2015. URL: <http://corporate.betfair.com/about-us/a-betting-revolution.aspx> (visited on 03/31/2016).
- [2] Betfair. *What is Betfair*. 2016. URL: <https://betting.betfair.com/what-is-betfair.html> (visited on 03/31/2016).
- [3] Jet Brains. *IntelliJ IDEA*. 2016. URL: <https://www.jetbrains.com/idea/> (visited on 03/31/2016).
- [4] Leo Breiman. *Out-of-bag estimation*. Tech. rep. Citeseer, 1996.
- [5] Leo Breiman. “Random forests”. In: *Machine learning* 45.1 (2001), pp. 5–32.
- [6] Chao Chen, Andy Liaw, and Leo Breiman. “Using random forest to learn imbalanced data”. In: *University of California, Berkeley* (2004).
- [7] Jerry Chou. *How to integrate python into a scala stack to build realtime predictive models*. 2014. URL: <http://www.slideshare.net/JerryChou4/how-to-integrate-python-into-a-scala-stack-to-build-realtime-predictive-models-v2-nomanuscript> (visited on 03/31/2016).
- [8] Betfair community. *API NG Migration update*. 2014. URL: <http://community.betfair.com/service/go/thread/view/94166/30402691/api-ng-migration-update> (visited on 03/31/2016).
- [9] Stack Exchange. *Overfitting*. 2016. URL: <http://stats.stackexchange.com/questions/192007/what-measures-you-look-at-the-determine-over-fitting-in-linear-regression> (visited on 03/31/2016).
- [10] Peter Flach. *Machine Learning: The Art and Science of Algorithms That Make Sense of Data*. New York, NY, USA: Cambridge University Press, 2012. ISBN: 1107422221, 9781107422223.
- [11] Play Framework. *Play Framework*. 2015. URL: <https://www.playframework.com/> (visited on 03/31/2016).
- [12] Play framework. *ScalaJSON*. 2016. URL: <https://www.playframework.com/documentation/2.5.x/ScalaJson> (visited on 03/31/2016).
- [13] Scala Fx. *ScalaFx*. 2016. URL: <http://www.scalafx.org/> (visited on 03/31/2016).
- [14] Pierre Geurts, Damien Ernst, and Louis Wehenkel. “Extremely randomized trees”. In: *Machine learning* 63.1 (2006), pp. 3–42.
- [15] GIT. *GIT fast verison control*. 2016. URL: <https://git-scm.com> (visited on 03/31/2016).
- [16] Git. *Getting Started About Version Control*. 2016. URL: <https://git-scm.com/book/en/v2/Getting-Started-About-Version-Control> (visited on 03/31/2016).
- [17] Lightbend Inc. *Akka*. 2016. URL: <http://akka.io/> (visited on 03/31/2016).
- [18] Lightbend Inc. *Akka FSM*. 2016. URL: <http://doc.akka.io/docs/akka/2.4.2/scala/fsm.html#the-fsm-trait-and-object> (visited on 03/31/2016).
- [19] Lightbend Inc. *Akka Remoting*. 2016. URL: <http://doc.akka.io/docs/akka/snapshot/scala/remoting.html> (visited on 03/31/2016).

- [20] Lightbend Inc. *Akka Round Robin Router*. 2016. URL: http://doc.akka.io/docs/akka/snapshot/scala/routing.html#RoundRobinPool_and_RoundRobinGroup (visited on 03/31/2016).
- [21] Typesafe Inc. *Spray*. 2016. URL: <http://spray.io/> (visited on 03/31/2016).
- [22] INRIA. *SciKit-learn*. 2016. URL: <http://scikit-learn.org/stable/> (visited on 03/31/2016).
- [23] The Chartered Institute for IT. *BCS Code of Conduct*. 2016. URL: <http://www.bcs.org/category/6030> (visited on 03/29/2016).
- [24] Scala Js. *ScalaJs*. 2016. URL: <https://www.scala-js.org/> (visited on 03/31/2016).
- [25] Ecole Polytechnique Federale de Lausanne (EPFL). *Scala*. 2016. URL: <http://www.scala-lang.org/> (visited on 03/31/2016).
- [26] Cohn Mike. *Succeeding with Agile: software development using Scrum*. Vol. The Addison-Wesley signature series. Addison-Wesley, 2010.
- [27] John J Murphy. *Technical analysis of the financial markets: A comprehensive guide to trading methods and applications*. Penguin, 1999.
- [28] Andrew Ng. *Machine Learning*. 2016. URL: <http://online.stanford.edu/course/machine-learning-1> (visited on 03/31/2016).
- [29] Pivotal. *Rabbit MQ*. 2016. URL: <https://www.rabbitmq.com/> (visited on 03/31/2016).
- [30] Betfair Developer Program. *Application Keys*. 2016. URL: <https://api.developer.betfair.com/services/webapps/docs/display/1smk3cen4v3lu3yomq5qye0ni/Application+Keys> (visited on 03/31/2016).
- [31] Betfair Developer Program. *Betfair API Overview*. 2016. URL: <https://api.developer.betfair.com/services/webapps/docs/display/1smk3cen4v3lu3yomq5qye0ni/API+Overview> (visited on 03/31/2016).
- [32] Betfair Developer Program. *Betfair Service NG*. 2016. URL: <https://github.com/city81/betfair-service-ng> (visited on 03/31/2016).
- [33] Betfair Developer Program. *Betting Type Definitions*. 2016. URL: <https://api.developer.betfair.com/services/webapps/docs/display/1smk3cen4v3lu3yomq5qye0ni/Betting+Type+Definitions> (visited on 03/31/2016).
- [34] Betfair Developer Program. *List Market Book*. 2016. URL: <https://api.developer.betfair.com/services/webapps/docs/display/1smk3cen4v3lu3yomq5qye0ni/listMarketBook> (visited on 03/31/2016).
- [35] Betfair Developer Program. *Login and Session Management*. 2016. URL: <https://api.developer.betfair.com/services/webapps/docs/pages/viewpage.action?pageId=3834909> (visited on 03/31/2016).
- [36] Betfair Developer Program. *Market Data Request Limits*. 2016. URL: <https://api.developer.betfair.com/services/webapps/docs/display/1smk3cen4v3lu3yomq5qye0ni/Market+Data+Request+Limits> (visited on 03/31/2016).
- [37] Betfair Developer Program. *Navigation Data For Applications*. 2015. URL: <https://api.developer.betfair.com/services/webapps/docs/display/1smk3cen4v3lu3yomq5qye0ni/Navigation+Data+For+Applications> (visited on 03/31/2016).
- [38] Honest Betting Reviews. *Betting Exchanges Compared*. 2016. URL: <http://www.honestbettingreviews.com/betting-exchanges/> (visited on 03/31/2016).

- [39] Chart School. *Technical Indicators and Overlays*. 2016. URL: http://stockcharts.com/school/doku.php?id=chart_school:technical_indicators (visited on 03/31/2016).
- [40] Cyril Schoreels, Brian Logan, and Jonathan M Garibaldi. “Agent based genetic algorithm employing financial technical analysis for making trading decisions using historical equity market data”. In: *Intelligent Agent Technology, 2004.(IAT 2004). Proceedings. IEEE/WIC/ACM International Conference on*. IEEE. 2004, pp. 421–424.
- [41] Apache Spark. *Mllib ensembles*. 2016. URL: <http://spark.apache.org/docs/latest/mllib-ensembles.html> (visited on 03/31/2016).