FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Trading Agent Framework Using Plain Strategies & Machine Learning

João Pedro Araújo Santos

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: Ana Paula Rocha

Co-supervisor: Rui Gonçalves

July 18, 2014

A Trading Agent Framework Using Plain Strategies & Machine Learning

João Pedro Araújo Santos

Mestrado Integrado em Engenharia Informática e Computação

Approved in oral examination by the committee:

Chair: Doctor A. Augusto de Sousa

External Examiner: Doctor Luís Paulo Reis

Supervisor: Doctor Ana Paula Rocha

July 18, 2014

Resumo

O mundo das mercados de apostas desportivas (*trading*) está em constante crescimento e com isso as pessoas estão a tentar melhorar os resultados do seu trading usando agentes automáticos de *trading*. Em analogia com os mercados financeiros, as operações de compra e venda são substituídas por apostas a favor e contra (*Back* and *Lay* respetivamente).

Esta tese descreve uma framework para ser usada no desenvolvimento de agentes automáticos de *trading* nos mercados da Betfair, utilizando uma interface de programação escrita em Java. A Betfair processa mais de cinco milhões de transações diárias (como fazer uma aposta) que representa mais do que todas as trocas feitas nas bolsas de ações Europeias combinadas. A Betfair está disponível 24 horas por dia, 7 dias por semana. Neste trabalho foram desenvolvidos dois agentes de trading, *DealerAgent* e *HorseLayAgent*, de acordo com a framework supra mencionada.

Os agentes mencionados atuam nos mercados "Para Ganhar" em corridas de cavalos do Reino Unido. Usam estratégias planas em conjunto com métodos de machine learning para melhorar os seus resultados de lucro/perda. Os agentes desenvolvidos foram submetidos a testes de viabilidade usando dados dos mercados "Para Ganhar" de corridas de cavalos do mercado de apostas Betfair, de Janeiro, Fevereiro e Maço de 2014.

Abstract

The world of online sports betting exchange (trading) is growing every day and with that people are trying to improve their trading by using automated trading. In analogy to the financial markets the buy and sell operations are replaced by betting for and against (Back and Lay).

This thesis describes a framework to be used to develop automated trading agents at Betfair sports markets using a Java programming interface. Betfair processes more than five million transactions (such as placing a bet) every day which is more than all European stock exchanges combined. Betfair is available 24 hours a day 7 days a week. For this thesis were developed two trading agents, DealerAgent and HorseLayAgent, accordingly with the presented framework.

The agents mentioned above act on To Win horse racing markets in United Kingdom. They use plain strategies together with machine learning methods to improve the profit/loss results. The developed agents were submitted to viability tests using data from Betfair To Win horse racing markets from January, February and March of 2014.

Acknowledgements

I would like to acknowledge the tireless effort made by Eng. Rui Gonçalves that, with its endless knowledge about online trading, helped me reaching the end of this journey.

Also, Dra. Ana Paula Rocha had a very important role guiding me with her opinions and expertise in thesis development.

Finally, I have to thank to my parents, sister and friends that always encouraged me to pursue my goals and be what I've always wanted to be, an Engineer.

Contents

List of Figures

List of Tables

LIST OF TABLES

Abbreviations

API	Application Programming Interface
BF	Betfair
FEUP	Engineering Faculty of Oporto University
FPS	Updates of prices (frames) per second
GUI	Graphical User Interface
PL	Profit/Loss
SL	Stoploss
TS	Trailing Stop
WEKA	Waikato Environment for Knowledge Analysis
WOM	Weight of Money

ABBREVIATIONS

Glossary

Back	Bet in favour of a selection (runner).
Betting Exchange	Entity that trade contracts on future events [FVN10].
Greenup	Generate equal profit on all selections, regardless of the outcome [Sof05a].
In-Play	The event has already started (is live) and not finished yet.
Ladder	Graphical interface used to trade that gives the trader an in depth perspective of the market [Tra14b].
Lay	Bet against a selection (runner).
Liquidity	Quantity of money in circulation.
Machine Learning	A general inductive process that automatically builds an automatic text classifier by learning, from a set of preclassified documents, the characteristics of the categories of interest. [Seb02]
Plain Strategy	Strategy to be used for trading, by an agent, before implementing machine learning methods [Reb10]. A negative plain strategy is the one that has a predicted negative profit/loss results in long term. On the other hand, a positive one has a predicted positive profit/loss results.
Runner	One selection of the market.
Stoploss	Condition to minimize losses.
Stopprofit	Condition to establish a maximum profit.
Trading Mechanism	Buy and sell process after market forecasting.
Weight of Money	Quantity of money on each side (Back and Lay). Used to determine whether or not a price is going to lengthen or shorten. [Tra14a].

Chapter 1

Introduction

Any person who is interested in sports events certainly also has his own opinions about the results, formations, moments of the game amongst others. And since it is this way, why not monetize this opinions and try to turn them profitable? This is the reason why there is now the online better (also called punter) concept. These are people available to invest some of their own money to, hopefully, have some profit with it. Usually, the greater the risk the greater the profit.

Typically, a punter starts on a betting house like, the most known, Bwin [Lim14b], Betclic [Lim14a] amongst others. The bets made are called "simple bets" and are made against the house, where after an event analysis and the bet matched by the house, the better will have to wait until the end of the market (it could be football or tennis games, horse racing, etc.) to make a profit. What happens a lot of times with this kind of bets is that the punters can not predict events in a long term (e.g. final score of a game) and they can change their opinion during the game. Since they already placed a bet, they can only hope that the initial analysis is correct and that they profit from what they predicted even if it is against the event tendency.

This led to appearence of online trading. Being a trader is like being an investor in the stock markets, but instead of betting in favor of a company they bet in favor of a team, a player, a horse, etc. As in any stock market, the price of the entity is always changing and being controlled by all the traders working on that market. In this case if you bet in favor of a team, for example to win, it means that someone else believed that the same team wouldn't be able to win the game. In online trading, the bets are made against other traders instead of betting against the house as it happens in the "simple bets".

By 2004, Betfair [Lim14d] was the world's largest betting exchange, with more than 50,000 people placing bets each week on the many events the website features. Elections, major horse races, golf tournaments, and soccer matches inevitably trade more than 3,7 million Euros at a time. [DPSW05] It has the biggest traders base and it is where there is more money flowing than in any other betting exchange, like Betdaq [Int14].

In order to be a good trader in live events it is important to bo very fast, due to the fact that markets are always changing. It is also essential to use a good trading software as the ones described in 2.2 instead of a simple web interface, as it is the case of Betfair, where market updates are considerably slower. If the trader wants an even faster way to trade, with the advantage of doing it automatically, the solution is automatic trading strategies (therefore called agents). These agents represent the trader on markets since they react to market changes in the way that he predifined, with a stable and well-defined set of preferences and actions depending on the clarity of the market [TZ88].

1.1 Objectives

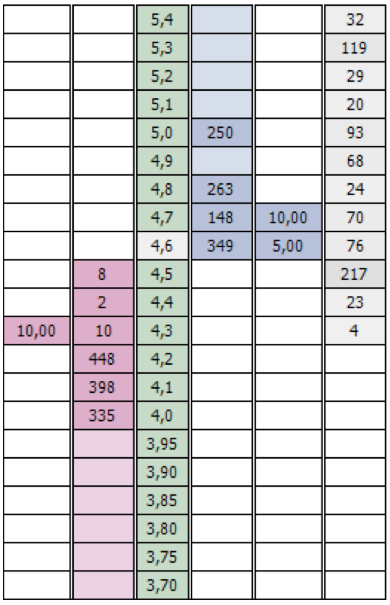
The main goal of this thesis is to present a framework (chapter3) suitable for develop trading agents. This framework enables trading agents to improve the profit/loss results, by combining plain strategies together with machine learning methods. As a proof of concept, two examples of trading agents in horse races, are implemented and presented during this work. They were developed using the high level interface called JBet (section 2.2.5). This programming interface is written in Java and eases the development process of an agent, eliminating the direct communication layer with Betfair API's at the same time that offers essential trading mechanisms that any good trader needs, such as trailing-stop, dutching, open trade, close trade, swing and scalping (section 2.1). The agents were submitted to several tests in different stages (defined on the framework) and both the results and conclusions are also presented in this thesis.

1.2 Problem Description

Betfair betting exchange includes markets, e.g. tennis match or horse race. On each market there are runners, e.g. horses in a horse race. On runners the Back/Lay bets are placed. Back bet means that the runner will win, Lay bet that it will lose. Fig. 1.1 illustrates the classical view of a runner in the market. In the middle column at green is the price scale. The price can also be referred as odd. Bets are placed on a given price, which represents the chances of the runner win. E.g. the price 2.0 is a 50% of chances (1/2 = 0.5), price 1.01 is a 99% of chances (1/1.01 = 0.99), 1000 is a 0.1% of chances (1/1000 = 0.001). On the left side, next to the prices column, in pink is the amount column formed by the Lay bets. On the right side, the column in blue, next to the prices column, is the amount column formed by the Back bets. The first and fifth column represent our own unmatched bets. In this case (Fig. 1.1) we have the following bets placed (but not matched yet):

- Lay of 10.00€ at 4.30 (Lay 10@4.3)
- Back of 10.00€ at 4.70 (Back 10@4.7)
- Back of 5.00€ at 4.60 (Back 5@4.6)

The last column on the right, in gray, is the volume column which represents the amount matched at every price of the ladder since the market was open. If a bet is placed at a price that "the market" is willing to buy, the bet will be matched at the best price offered. For example, in the market state of Fig. 1.1 if a Back bet 15@4.4 is placed (on the blue side) it will match 8.00@4.5 and 2.00@4.4 , and will leave the remaining 5.00@4.4 unmatched (waiting for someone to buy with a Lay bet). The traded volume information will be updated. This is how the prices move in the market. Since this bet was matched at two different prices, the global matched price of this bet can be calculated using equation 1.1.



		5,4			32
		5,3			119
		5,2			29
		5,1			20
		5,0	250		93
		4,9			68
		4,8	263		24
		4,7	148	10,00	70
		4,6	349	5,00	76
	8	4,5			217
	2	4,4			23
10,00	10	4,3			4
	448	4,2			
	398	4,1			
	335	4,0			
		3,95			
		3,90			
		3,85			
		3,80			
		3,75			
		3,70			

Figure 1.1: Ladder data example of a runner in the market [Lim14c]

$$PriceAverage = \frac{\sum_{n=1}^{n}(Price_n \times Amount_n)}{\sum_{n=1}^{n}(Amount_n)} \tag{1.1}$$

If a Back is placed above the best offer in the market (4.5 in Fig. 1.1), for example 15@4.9, it will stay in the market unmatched and, so for, waiting to be matched. The same happens to a Lay bet if it is placed at a lower price than the best offer (counter bet waiting to be matched). Only unmatched or partial unmatched bets can be canceled. The profit of a Back bet is calculated using equation 1.2 and the liability (in case of loss) of a Back bet is the amount of the bet itself.

$$ProfitBackBet = AmountBack \times (PriceBack - 1) \tag{1.2}$$

The liability or amount in case of loss of a Lay bet is given by equation 1.3 and the profit is the amount of the bet itself. Basically the Lay is the "mirror" of Back.

$$LiabilityLayBet = AmountLay \times (PriceLay - 1) \tag{1.3}$$

Using combinations of Back/Lay it is possible to assure profit (or loss) before the end result of an event, using the price movement on the runner (odd variation). Example of a trade where it is not need to know end result of an event to have secure profit:

- Back of 2.00e at 2.12 (Back 2@2.12) Matched
- Lay of 2.00e at 2.10 (Lay 2@2.1) Matched

For a bet to be matched it must become the best offer in the market and it has to be purchased with a counter bet. When the runner is a winner, then the profit (Back bet) – loss (Lay bet) is:

2 × (2.12 - 1) – 2 × (2:10 - 1) = 2.24 × 2.20 = 0.04

When the runner is a looser, then the profit (Lay bet) – loss (Back bet) is:

2–2=0

Notice that if we have this kind of Back/Lay bet combination, on the same runner with the same amount at different prices, there will be profit (if the Back price is higher than the Lay price) or loss (if the Back price is lower than the Lay price) only if the runner in question wins the event. If any other runner wins the event and the combination of a Back/Lay bets have the same amount, the profit/loss will be 0. To distribute the profit/loss equal for all runners (outcomes) the amount of the bet used to close the trade must be recalculated. This process is called "do the greening" or "hedging". If a Back position is open on the market, the amount to close the position with the corresponded Lay bet is calculated using equation 1.4.

$$CloseAmountLay = \frac{PriceOpenInBack}{PriceLayToClose} \times AmountOpenInBack \tag{1.4}$$

If a Lay position is open on the market, the amount to close the position with the corresponded Back is calculated using equation 1.5.

$$CloseAmountBack = \frac{PriceOpenInLay}{PriceBackToClose} \times AmountOpenInLay \tag{1.5}$$

1.3 Structure

Besides the introduction, this report has 4 more chapters. In chapter 2 the state of the art is approached with descriptions of trading tools, trading mechanisms and machine learning packages wich can be applied with the framework. Chapter 3 describes the proposed trading agent framework, enhancing its main modules. In chapter 4 it is shown the implementation of two trading agents in the horse racing scenario and in chapter 5 it is presented the tests made and results that these two trading agents achieved. Finally, in chapter 6 are presented some conclusions and pointed out directions to pursue in future work.

Chapter 2

State of the art

In this chapter are presented, in 2.1, some well known trading mechanisms used by traders. In 2.2 are presented some existing tools that improve trading speeds at the same time that make available some of the trading mechanisms of 2.1. Finally, in 2.3, are presented and explained the machine learning packages wich can be used during the implementation of this work.

2.1 Trading Mechanisms

Traders are constantly using trading mechanisms that represents a buy and sell process after market forecasting. The most common ones are Scalping, Swing, Trailing Stop and Dutching. The next sections will describe all of the already existent trading mechanisms and a new one with name Dealer.

2.1.1 Scalping

Scalping is short term trading. A scalping trader looks to make lots of small profits, which in time add up. Scalping relies on lots of active participants in the market. Scalping works better in markets with high liquidity. The concept is simple, if a back bet is matched at a certain price, a lay bet must be placed right in the next lower price, or, if a lay bet is matched at a price a back bet must be placed right in the next higher price in the ladder to make profit. The profit/loss is equal to the difference, or spread, between the Back and Lay price. The Betfair betting exchange is an ideal place to trade in this way. Mainly in horse racing because there is high liquidity being matched in these markets, in particular just before the start of the race. Scalping the market means trading in the market tick by tick. One tick is one step in the prices scale of the ladder. For example if a Back at 2.12 is placed, one successful scalp will close the position with Lay at 2.10 (one tick down). If a trade starts with a back it means that the price was forecast to go down. If it is predicted to go up the scalp starts with a Lay bet.

Figure 2.1 represents the state machine used to process one Back->Lay scalping (prediction for the prices to go down). One Lay->Back scalp will be a "mirror" of this state machine. The Price Back Request- PBR is the price where the agent enters the market. If the prices already

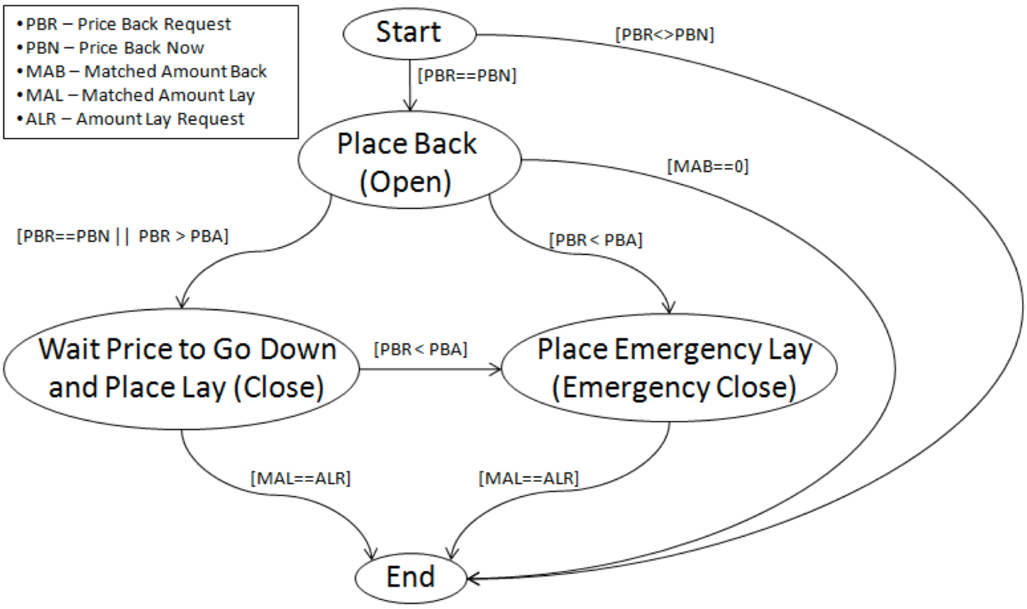

• PBR – Price Back Request
• PBN – Price Back Now
• MAB – Matched Amount Back
• MAL – Matched Amount Lay
• ALR – Amount Lay Request
Start
[PBR<>PBN]
[PBR==PBN]
Place Back (Open)
[MAB==0]
[PBR==PBN || PBR > PBA]
[PBR < PBA]
Wait Price to Go Down and Place Lay (Close)
[PBR < PBA]
Place Emergency Lay (Emergency Close)
[MAL==ALR]
[MAL==ALR]
End

Figure 2.1: Simplified graph schema for a Back->Lay scalp implementation. [RGP13]

moved [PBR <> PBN] when the order reach the scalp module (the start state), it will assume the opportunity was lost (the prices already went down) or the prices went in the wrong predicted direction (up) so it ends the process without doing nothing. Otherwise [PBR == PBN], opens position on the market with a Back bet. After the bet is ordered to be placed, if the bet was not matched it will end the trade (canceling the bet) because it will assume the prices already move down, while the order was reaching the Betfair server, and the opportunity was lost. Otherwise it will try to close the position placing a Lay bet. If the price goes one tick down it will close the trade with profit. If the price does not move it will wait. If the price goes up it will close in "emergency" with loss.[RGP13]

2.1.2 Swing

The swing mechanism is very similar to the scalping. The main difference is the number of ticks the prices have to move in order to enter the close state (profit or loss close). On the swing mechanism it is possible to define the offset number of ticks to close in profit and the offset number of ticks to close in loss. If the prices stay inside this interval offset (up and down) it does nothing. Swing with offset of 1 tick for profit and offset of 1 tick for loss is the same as scalping. [RGP13]

2.1.3 Trailing Stop

The trailing-stop mechanism is used when the agent is looking to catch a much broader trend in a market but wants to retain a stop loss condition if the trend starts to turn. After a position is open in the market (open bet is matched), the close bet is placed with a tick offset behind, and moves only when the price moves in the predicted direction. Eventually the price will move in the

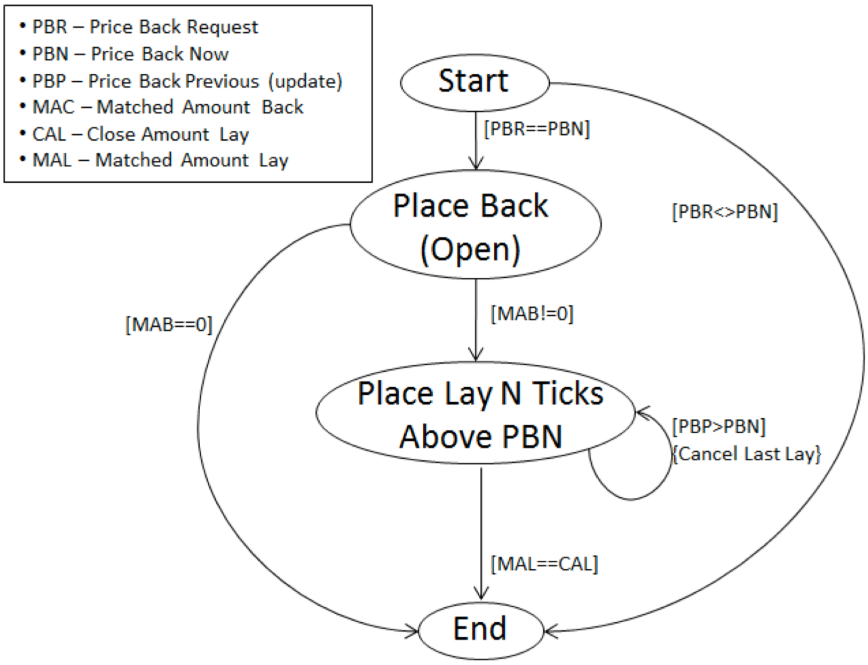


Figure 2.2: Simplified graph schema for a Back->Lay Trailing-Stop implementation. [RGP13]

reverse direction reaching the close bet. Figure 2.2 represents the state machine used to process this method, for the price prediction to go down (Back to open -> Lay to close). The state "Place Lay N Ticks Above PBN", being N the tick offset to follow the price, is repeated when [PBP > PBN] the runner price moves in the predicted direction (down). If [MAL == CAL] the close bet is completely matched, it means the price went in the reverse direction (up) and reach the close bet, closing the trade. [RGP13]

2.1.4 Dutching

Dutching consists in backing several selections in an event so that no matter which wins, the returns are the same. Dutching is a method used to divide stake over a number of selections in an event so that the same amount is won irregardless of which selection wins. This technique is useful when there are two or more outcomes you wish to back and have a specific amount to stake. Before the rise of betting exchanges, dutching was sometimes used as an elaborate way to lay (bet against) a selection by backing everything else. [Ace13]

2.1.5 Dealer

The dealer mechanism, represented in figure 2.3, consists in backing and laying at the same time and waiting that both positions are matched returning a profit. The tick difference between back and lay is always defined by the trader. The higher this offset the higer are the profits. As a security measure, there are two stoploss possibilities (one for back and other for lay) that will enter in action as soon as the market odd reaches any of these stoploss points. At this moment, the remaining unmatched position will be canceled and there will be an attempt to minimize loss closing the position with remaining unmatched value at current market best price. This technique has best results in markets with a lot of uncertainty that become very volatile.

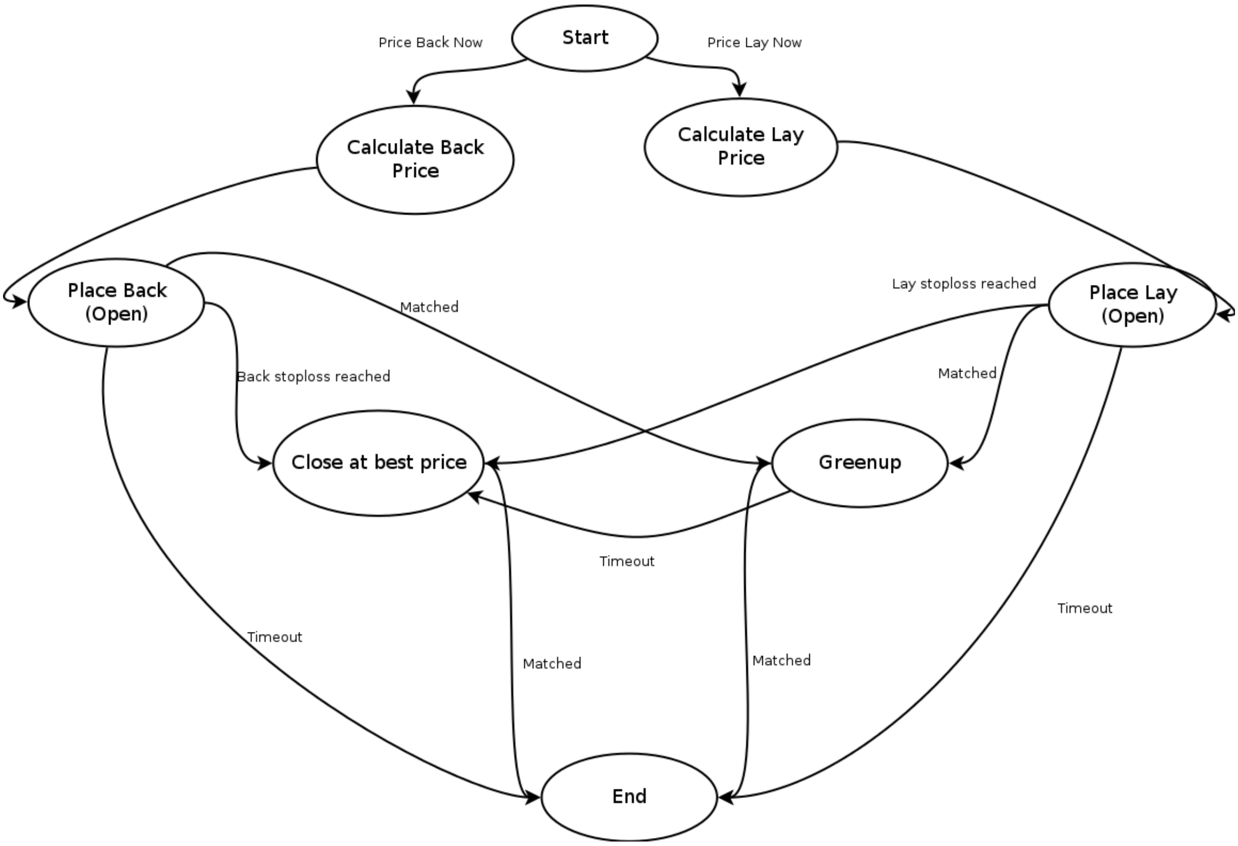


Figure 2.3: Schematic representation of Dealer mechanism

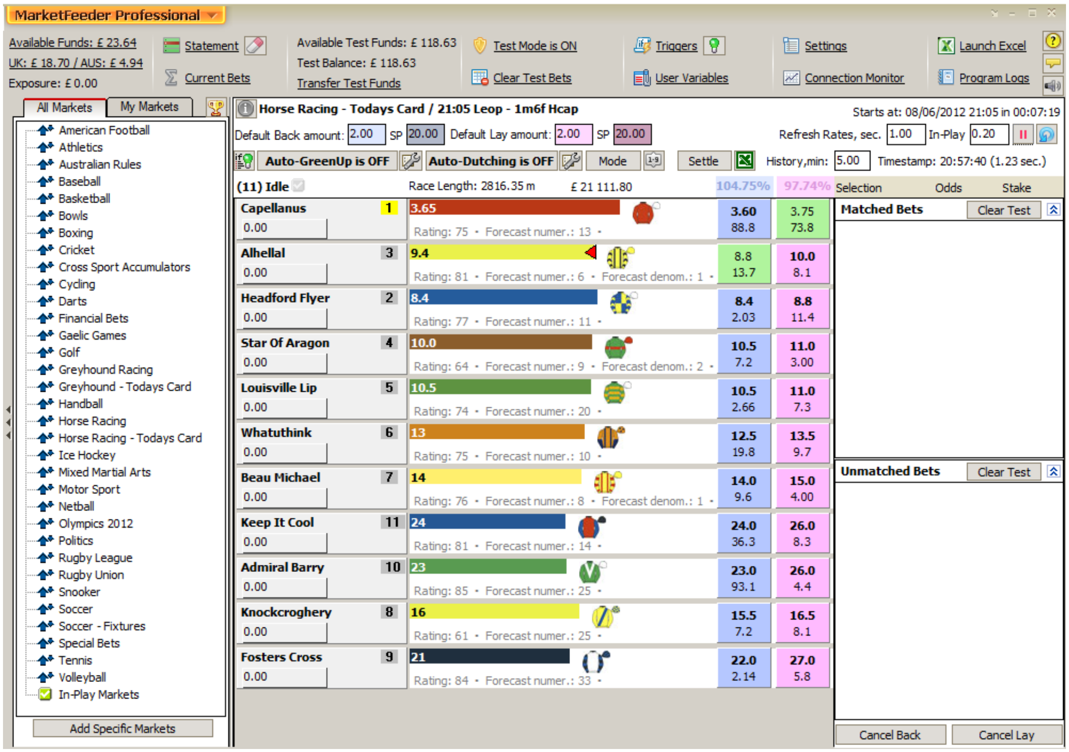
2.2 Trading Tools

We used the term trading tools to refer software utillities that allow users to create transactios at betting exchanges. This section describes some of the trading tools most used by the community of online trading.

2.2.1 Market Feeder Professional

MarketFeeder Pro is a betting application for BetFair oriented for automated trading. It combines all the usual tools a BetFair bettor needs with unique applications for scheduled, preprogrammed tasks. [Sof05b] Its main functionalities are:

- Test mode
- Update interval up to 0.3 seconds



Available Funds: £ 23.64
UK: £ 18.70 / AUS: £ 4.94
Exposure: £ 0.00

Available Test Funds: £ 118.63
Test Balance: £ 118.63
Transfer Test Funds

Horse Racing - Todays Card / 21:05 Leop - 1m6f Hcap

Starts at: 08/06/2012 21:05 in 00:07:19

Default Back amount: 2.00 SP 20.00 Default Lay amount: 2.00 SP 20.00

Refresh Rates, sec. 1.00 In-Play 0.20

History,min: 5.00 Timestamp: 20:57:40 (1.23 sec.)

(11) Idle Race Length: 2816.35 m £ 21 111.80

				104.75%	97.74%
Capellanus 0.00	1	3.65	Rating: 75 • Forecast numer.: 13 •	3.60 88.8	3.75 73.8
Alhellal 0.00	3	9.4	Rating: 81 • Forecast numer.: 6 • Forecast denom.: 1 •	8.8 13.7	10.0 8.1
Headford Flyer 0.00	2	8.4	Rating: 77 • Forecast numer.: 11 •	8.4 2.03	8.8 11.4
Star Of Aragon 0.00	4	10.0	Rating: 64 • Forecast numer.: 9 • Forecast denom.: 2 •	10.5 7.2	11.0 3.00
Louisville Lip 0.00	5	10.5	Rating: 74 • Forecast numer.: 20 •	10.5 2.66	11.0 7.3
Whatuthink 0.00	6	13	Rating: 75 • Forecast numer.: 10 •	12.5 19.8	13.5 9.7
Beau Michael 0.00	7	14	Rating: 76 • Forecast numer.: 8 • Forecast denom.: 1 •	14.0 9.6	15.0 4.00
Keep It Cool 0.00	11	24	Rating: 81 • Forecast numer.: 14 •	24.0 36.3	26.0 8.3
Admiral Barry 0.00	10	23	Rating: 85 • Forecast numer.: 25 •	23.0 93.1	26.0 4.4
Knockcroghery 0.00	8	16	Rating: 61 • Forecast numer.: 25 •	15.5 7.2	16.5 8.1
Fosters Cross 0.00	9	21	Rating: 84 • Forecast numer.: 33 •	22.0 2.14	27.0 5.8

Selection	Odds	Stake

Figure 2.4: Guided user interface of MarketFeeder Professional [Sof05b]

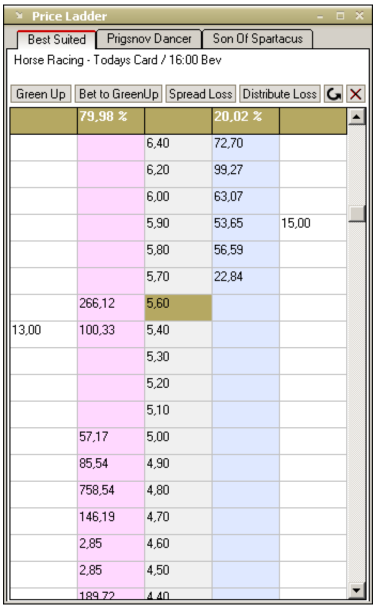
- Auto-greenup
- Auto-dutching
- Ladder

The test mode alows users to define a ficitious amount for its betting account and utilize it for making simulation bets. This is useful for testing a new plain strategy whose efficiency is not yet known.

At online betting and trading, specially on live markets, is extremly important to have the least possible refreshing rates. The 0.3 seconds update interval offered by MarketFeeder Pro is enough to users have the odds in synchronization with the market.

The auto-greening up (close all positions in market) functionality is a basic yet essential one that most traders use constantly. Even Betfair introduced this functionality on its web interface under the name of Cash Out[Lim13]. It allows traders to automatically greenup, or in other words, generate equal profit on all selections, regardless of the outcome [Sof05a]. For exemple, if a user opens a back position at 1.10 with a stake of 10.00€ and choose to greenup when the odd of that market is at 1.09 he will have a profit of one tick resulting in 0.09€.

In Figure 2.4 it is possible to see how MarketFeeder Pro interface looks like. In its most basic form it is very similar to Betfair web interface but with lower refresh rates. It includes a market navigator on the left side, market contents on centre with possible selections and odd values (blue



	79,98 %		20,02 %	
		6,40	72,70	
		6,20	99,27	
		6,00	63,07	
		5,90	53,65	15,00
		5,80	56,59	
		5,70	22,84	
	266,12	5,60		
13,00	100,33	5,40		
		5,30		
		5,20		
		5,10		
	57,17	5,00		
	85,54	4,90		
	758,54	4,80		
	146,19	4,70		
	2,85	4,60		
	2,85	4,50		
	189,72	4,40		

Figure 2.5: Ladder interface of MarketFeeder Professional [Sof05b]

to back a selection and pink to lay selection) and configuration options both of the market and software at top of the window.

In Figure 2.5 is presented the ladder interface of MarketFeeder Pro. This is the most commonly graphical interface used by traders, as it allows to have, in only one place, how much money is currently waiting to be matched on each odd, what is the odd value for back and lay at that moment and the weight of money.

2.2.2 Gruss Software

Gruss Software is an alternative software to the Betfair site. It allows traders to place bets more efficiently and develop their own strategies.[Sof10]

This software allows its users to:

- Greening
- Dutching
- Use a ladder
- Bet with tick offset with stop loss
- Trigger bet from an Excel spreadsheet

GrussBet is probably the most basic software here described because it has the minimum functionalities needed for successful trading. It is also the cheapest software, so it is ideal for any rookie that is starting in the trading world not having to spend a lot of money even before start winning it at trading.

It has a programming interface through Excel that allows the trader to create triggered bets accordingly to market values. These values are copied into a Excel spreadsheet one time at each second and using Excel capabilities is possible to bet if some odds hit any designated value or use any other kind of conditions with these copied values. For any rookie is an ideal tool to start entering the world of automated trading but its flexibility is limited.

2.2.3 Bet Angel

Bet Angel is probably the most used software by professional traders at Betfair.

Sports trading began getting popular in 2004. Back then, Betfair was the leading betting exchange. Whoever wanted to trade sports and buy or sell betting odds, had to register an account with Betfair. As sports bettors gradually converted into sports traders, it became apparent that they would need help to maintain their edge against the fierce opposition. [Mak14]

That help came in the form of speed. Traders need speed to be successful in trading, and especially if they are scalping. [Mak14] This speed enables traders to work on live markets like football or horse races and always be in synchronization with market giving them a tremendous advantage over traders that simply use the web interface of Betfair that refreshes its prices at a lower rate than Bet Angel does.

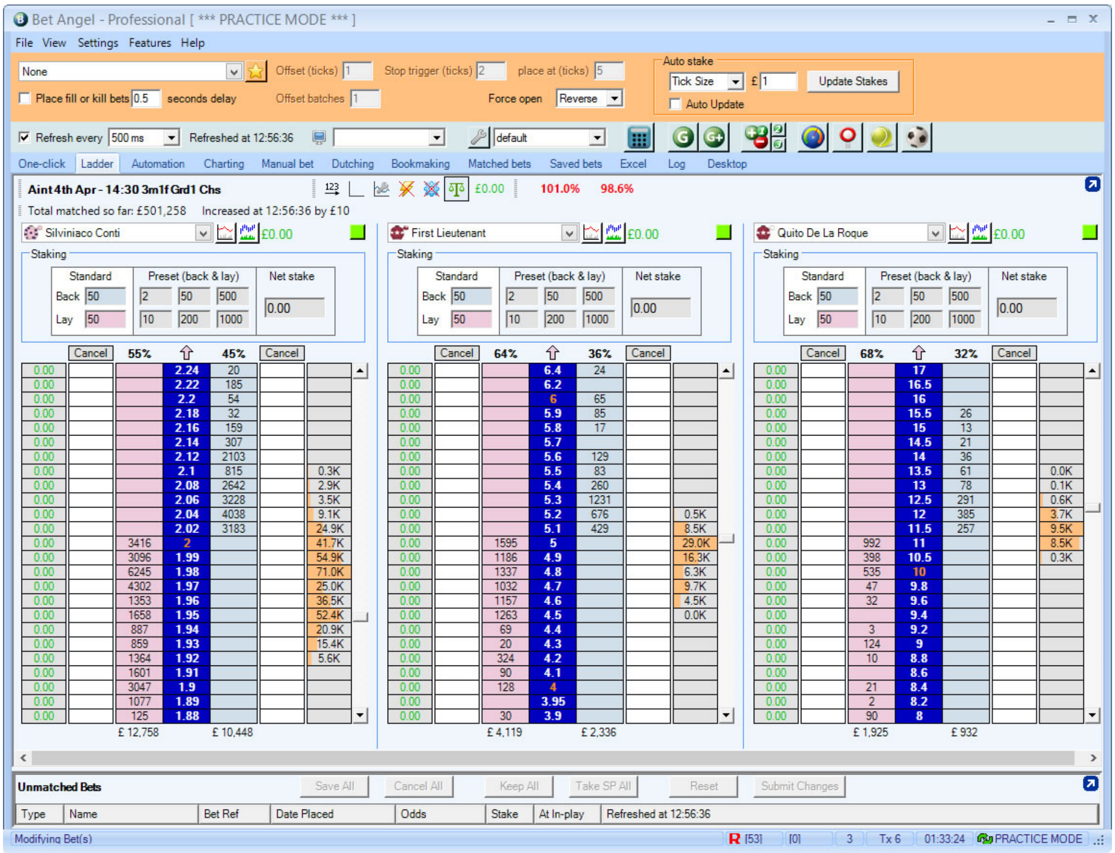
Bet Angel main features are:

- Practice mode
- Dutching & Bookmaking
- Excel spreadsheet integration
- Cash-out
- Tennis mathematical model
- Football mathematical model

What really differentiates Bet Angel from other trading softwares is in first place its user interface that is accessible to everyone but at the same time has everything on its place making it easy and fast to reach every functionality. The second main functionality is to have time a mathematical model to be applied on tennis and soccer. These are separate mathematical models for odd prediction over time during in play events. As the names suggest, Soccer Mystic is for soccer markets and Tennis trader for tennis markets.

Figures 2.6 and 2.7 show how the ladder user interface, repectively, and dutching user interface are presented to traders. It is possible to open multiple ladders that can be of extreme importance on markets like horse racing allowing to open one ladder for each runner we are interested trading in. Dutching interface is also very simple but extremely powerful as with just clicking the selections we want to dutch in, is possible to make a complete dutch bet without have to worry with manual calculations of odds and stakes.

Both tennis and and soccer mathematical models offer traders a set of data and analysis that otherwise is impossible to get by just looking at the markets. Both offer a odd prediction model that tries to predict where odds will be if some events occur on the game. This way is possible to better assess the risk that trader is assuming even before making the bet. Of course this is only an indicative tool because is a prediction that may not be right. It is more of a guiding tool to help traders. Figures 2.8 and 2.9 present how the user interfaces for Soccer Mystic and Tennis trader are respectivly. Figure 2.8 is possible to see the odd prediction of match odds market when the Home team (Manchester United) scores the first goal of the game at minute 20th. It is possible to simulate up to 6 goals and see odd predictions on match odds, over/under, correct score and total goals markets. Figure 2.9 shows the functionality that enables a trader to check the probability that each of the players have to win the match (altough, it is possible to check the probability to win only the set). The column "Predicted Odds" ipresents the most interesting information as it tells the trader what the odds will be if any of the events described in the rows happens. This way the trader can focus his energies on trying to predict what will happen in the game, leaving the odd prediction to Bet Angel with this Tennis trader.



None
Offset (ticks) 1
Stop trigger (ticks) 2
place at (ticks) 5
Auto stake: Tick Size £ 1
□ Auto Update
□ Place fill or kill bets 0.5 seconds delay
Offset batches 1
Force open Reverse
☑ Refresh every 500 ms
Refreshed at 12:56:36
default

Aint 4th Apr - 14:30 3m1f Grd1 Chs
£0.00 101.0% 98.6%
Total matched so far: £501,258 Increased at 12:56:36 by £10

Silviniaco Conti £0.00

Staking

Standard		Preset (back & lay)			Net stake
Back	50	2	50	500	0.00
Lay	50	10	200	1000	

	Cancel	55%	⇧	45%	Cancel	
0.00			2.24	20		
0.00			2.22	185		
0.00			2.2	54		
0.00			2.18	32		
0.00			2.16	159		
0.00			2.14	307		
0.00			2.12	2103		
0.00			2.1	815		0.3K
0.00			2.08	2642		2.9K
0.00			2.06	3228		3.5K
0.00			2.04	4038		9.1K
0.00			2.02	3183		24.9K
0.00		3416	2			41.7K
0.00		3096	1.99			54.9K
0.00		6245	1.98			71.0K
0.00		4302	1.97			25.0K
0.00		1353	1.96			36.5K
0.00		1658	1.95			52.4K
0.00		887	1.94			20.9K
0.00		859	1.93			15.4K
0.00		1364	1.92			5.6K
0.00		1601	1.91			
0.00		3047	1.9			
0.00		1077	1.89			
0.00		125	1.88			
		£ 12,758		£ 10,448		

First Lieutenant £0.00

Staking

Standard		Preset (back & lay)			Net stake
Back	50	2	50	500	0.00
Lay	50	10	200	1000	

	Cancel	64%	⇧	36%	Cancel	
0.00			6.4	24		
0.00			6.2			
0.00			6	65		
0.00			5.9	85		
0.00			5.8	17		
0.00			5.7			
0.00			5.6	129		
0.00			5.5	83		
0.00			5.4	260		
0.00			5.3	1231		
0.00			5.2	676		0.5K
0.00			5.1	429		8.5K
0.00		1595	5			29.0K
0.00		1186	4.9			16.3K
0.00		1337	4.8			6.3K
0.00		1032	4.7			9.7K
0.00		1157	4.6			4.5K
0.00		1263	4.5			0.0K
0.00		69	4.4			
0.00		20	4.3			
0.00		324	4.2			
0.00		90	4.1			
0.00		128	4			
0.00			3.95			
0.00		30	3.9			
		£ 4,119		£ 2,336		

Quito De La Roque £0.00

Staking

Standard		Preset (back & lay)			Net stake
Back	50	2	50	500	0.00
Lay	50	10	200	1000	

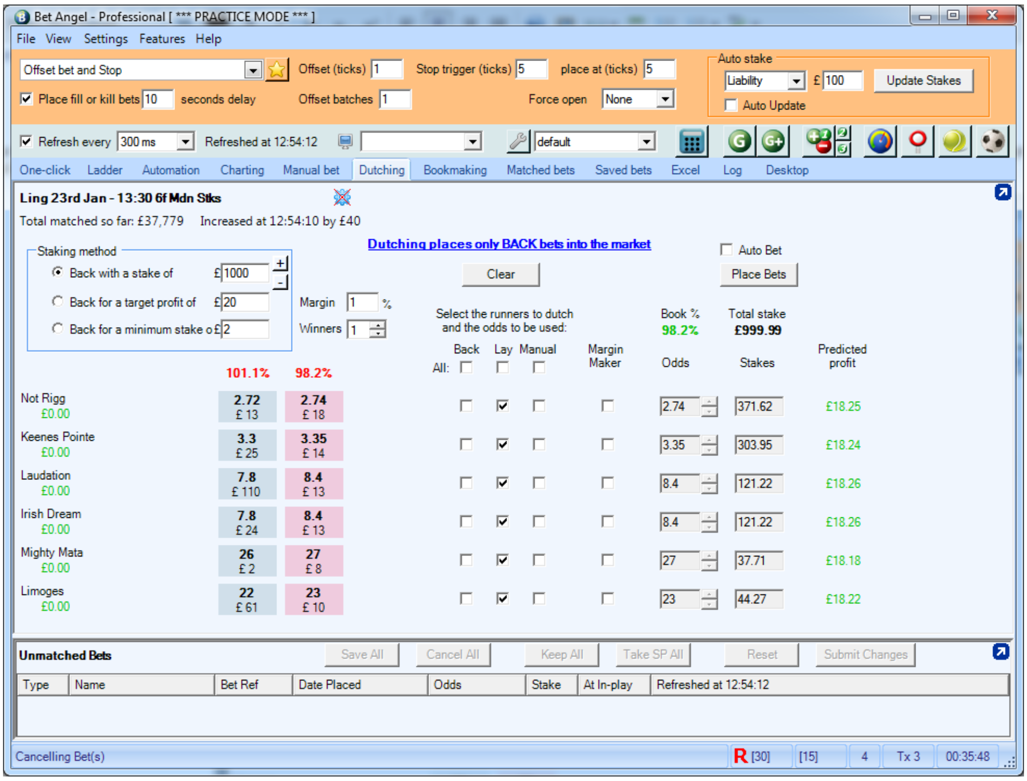
	Cancel	68%	⇧	32%	Cancel	
0.00			17			
0.00			16.5			
0.00			16			
0.00			15.5	26		
0.00			15	13		
0.00			14.5	21		
0.00			14	36		
0.00			13.5	61		0.0K
0.00			13	78		0.1K
0.00			12.5	291		0.6K
0.00			12	385		3.7K
0.00			11.5	257		9.5K
0.00		992	11			8.5K
0.00		398	10.5			0.3K
0.00		535	10			
0.00		47	9.8			
0.00		32	9.6			
0.00			9.4			
0.00		3	9.2			
0.00		124	9			
0.00		10	8.8			
0.00			8.6			
0.00		21	8.4			
0.00		2	8.2			
0.00		90	8			
		£ 1,925		£ 932		

Unmatched Bets

Type	Name	Bet Ref	Date Placed	Odds	Stake	At In-play	Refreshed at 12:56:36

Modifying Bet(s) R [53] [0] 3 Tx 6 01:33:24 PRACTICE MODE

Figure 2.6: Ladder interface of Bet Angel [Ang14]



Bet Angel - Professional [*** PRACTICE MODE ***]

Offset bet and Stop   Offset (ticks) 1   Stop trigger (ticks) 5   place at (ticks) 5

☑ Place fill or kill bets 10 seconds delay   Offset batches 1   Force open None

Auto stake: Liability £ 100   □ Auto Update

☑ Refresh every 300 ms   Refreshed at 12:54:12   default

Ling 23rd Jan - 13:30 6f Mdn Stks

Total matched so far: £37,779   Increased at 12:54:10 by £40

Dutching places only BACK bets into the market

□ Auto Bet

Staking method
- ● Back with a stake of £1000
- ○ Back for a target profit of £20
- ○ Back for a minimum stake o £2

Margin 1 %   Winners 1

Select the runners to dutch and the odds to be used: Book % 98.2%   Total stake £999.99

	101.1%	98.2%	Back	Lay	Manual	Margin Maker	Odds	Stakes	Predicted profit
All:			□	□	□				
Not Rigg £0.00	2.72 £ 13	2.74 £ 18	□	☑	□	□	2.74	371.62	£18.25
Keenes Pointe £0.00	3.3 £ 25	3.35 £ 14	□	☑	□	□	3.35	303.95	£18.24
Laudation £0.00	7.8 £ 110	8.4 £ 13	□	☑	□	□	8.4	121.22	£18.26
Irish Dream £0.00	7.8 £ 24	8.4 £ 13	□	☑	□	□	8.4	121.22	£18.26
Mighty Mata £0.00	26 £ 2	27 £ 8	□	☑	□	□	27	37.71	£18.18
Limoges £0.00	22 £ 61	23 £ 10	□	☑	□	□	23	44.27	£18.22

Unmatched Bets

Type	Name	Bet Ref	Date Placed	Odds	Stake	At In-play	Refreshed at 12:54:12

Cancelling Bet(s)   R [30]   [15]   4   Tx 3   00:35:48

Figure 2.7: Dutching interface of Bet Angel [Ang14]

2.2.4 BFExplorer

The BFExplorer is not only a betting or trading platform for betfair, but also the platform that can be used to automate betting and trading systems. [Bel14a]

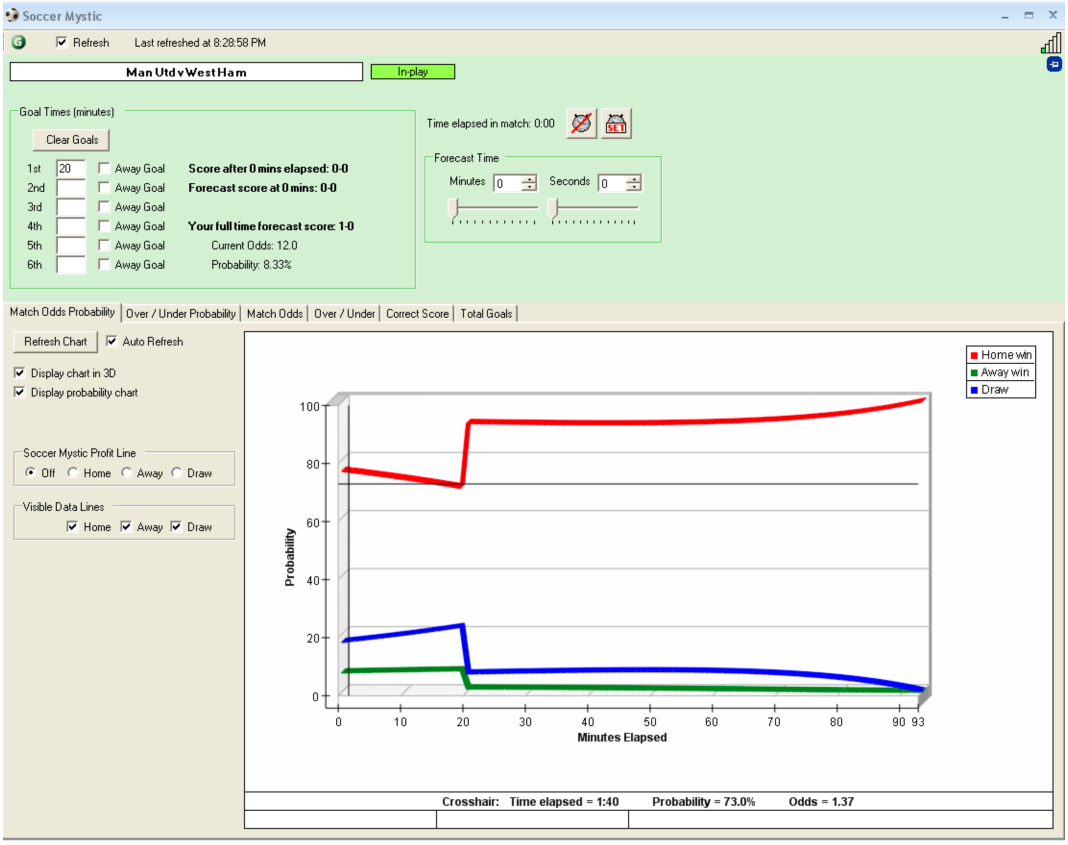
The Bfexplorer PRO offers three types of user interface for bet placing, the market grid view similar to the user interface on betfair web pages (Figure 2.10), the selection list view ("ladder") user interface for placing, updating and cancelling bets (Figure 2.11), and BFExplorer Trader for multi selection trading on the list view interface (Figure 2.11).

After a bet is placed at betfair market, the selection where the bet was made is automatically added into the Watched selections window where it is possible to see the price/odds movement and traders' current position. If the trader want to close the position (trade out with equal profit), simply click on Profit button, or on the Close position in the Watched Selections window.

It is possible to monitor as many markets as the trader want, quickly switching among them through Watched Selection window or Market list.

The application offers a set of Bet Wizards for placing bets when preset criteria are met, bet wizards for placing a bet and closing position, backing or laying all selections, back or lay dutching.

As this application is used by traders, it offers trading strategies for any of mentioned user interface components and extensive bot script support both by built-in bots or customizable bot scripts. This unique feature allows the trader to write their own bot scripts in Visual Basic or



Soccer Mystic

☑ Refresh Last refreshed at 8:28:58 PM

Man Utd v West Ham In-play

Goal Times (minutes)

1st 20 ☐ Away Goal
2nd ☐ Away Goal
3rd ☐ Away Goal
4th ☐ Away Goal
5th ☐ Away Goal
6th ☐ Away Goal

Score after 0 mins elapsed: 0-0
Forecast score at 0 mins: 0-0

Your full time forecast score: 1-0
Current Odds: 12.0
Probability: 8.33%

Time elapsed in match: 0:00

Forecast Time
Minutes 0 Seconds 0

☑ Auto Refresh
☑ Display chart in 3D
☑ Display probability chart

Soccer Mystic Profit Line
● Off ○ Home ○ Away ○ Draw

Visible Data Lines
☑ Home ☑ Away ☑ Draw

Crosshair: Time elapsed = 1:40 Probability = 73.0% Odds = 1.37

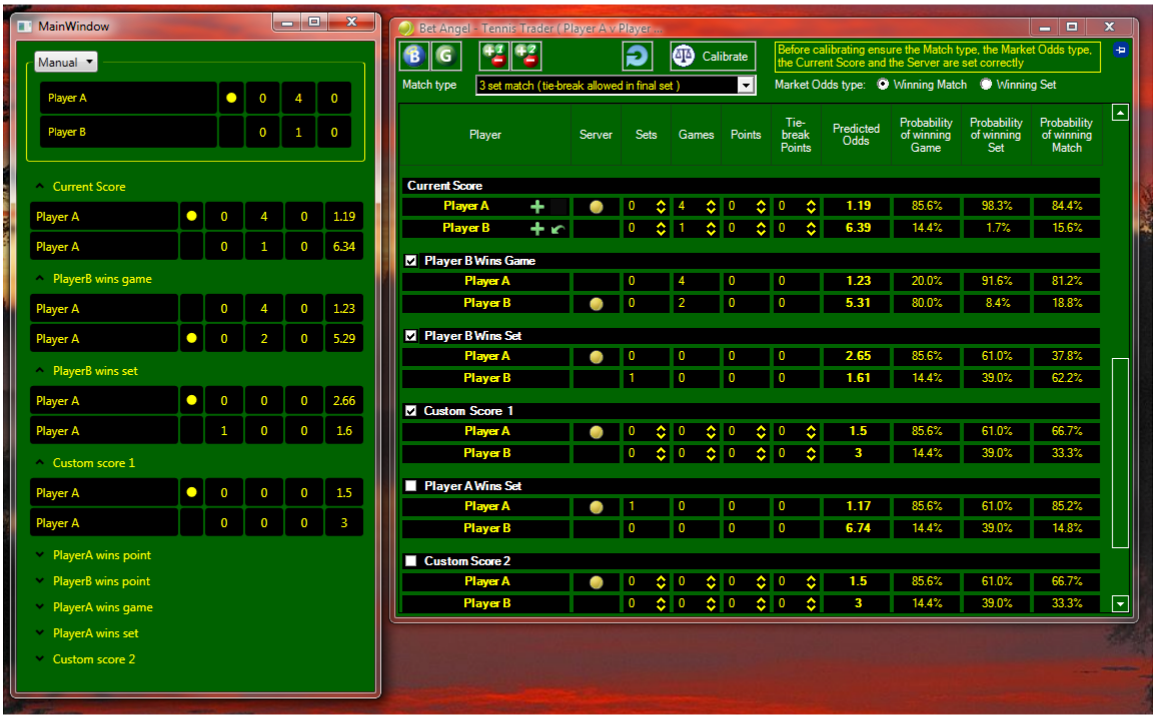
Figure 2.8: Soccer Mystic interface of Bet Angel [Tra12]

C# programming languages. For users who are not familiar with any programming language the Bfexplorer offers a Bot Executor with Bot Criteria Editor which can be used to setup different scenarios for bet placing or trading by utilizing all offered bots (including customize bots). Such bot criteria is then used on the selection when needed, just by clicking on the start bot toolbar button or for fully automated solutions which can run on the Trade Opportunity Lookup service. The application supports the Excel automation as well. Bfexplorer offers any available information the trader can get from betfair, presenting the market data on charts where is possible to see the price/odds movement and weight of money in time, traded and available volume and market statistics. [Bel14b]

2.2.5 JBet

JBet is a state of the art Java software framework developed at Department of Informatics Engineering in School of Engineering, Porto University (FEUP) used to create trading agents for betfair.

JBet is the base software on wich this work was developed. It is a complex and powerful set of function calls to Betfair API in Java. JBet allows developers to interact with Betfair in a higher level making possible to develop agents without too much effort. JBet has already some trading mechanisms implemented. Each trading mechanism inside JBet is a set of predefined actions that



MainWindow

Manual

Player A	●	0	4	0
Player B		0	1	0

Current Score

Player A	●	0	4	0	1.19
Player A		0	1	0	6.34

PlayerB wins game

Player A		0	4	0	1.23
Player A	●	0	2	0	5.29

PlayerB wins set

Player A	●	0	0	0	2.66
Player A		1	0	0	1.6

Custom score 1

Player A	●	0	0	0	1.5
Player A		0	0	0	3

PlayerA wins point

PlayerB wins point

PlayerA wins game

PlayerA wins set

Custom score 2

Bet Angel - Tennis Trader (Player A v Player ...

Calibrate

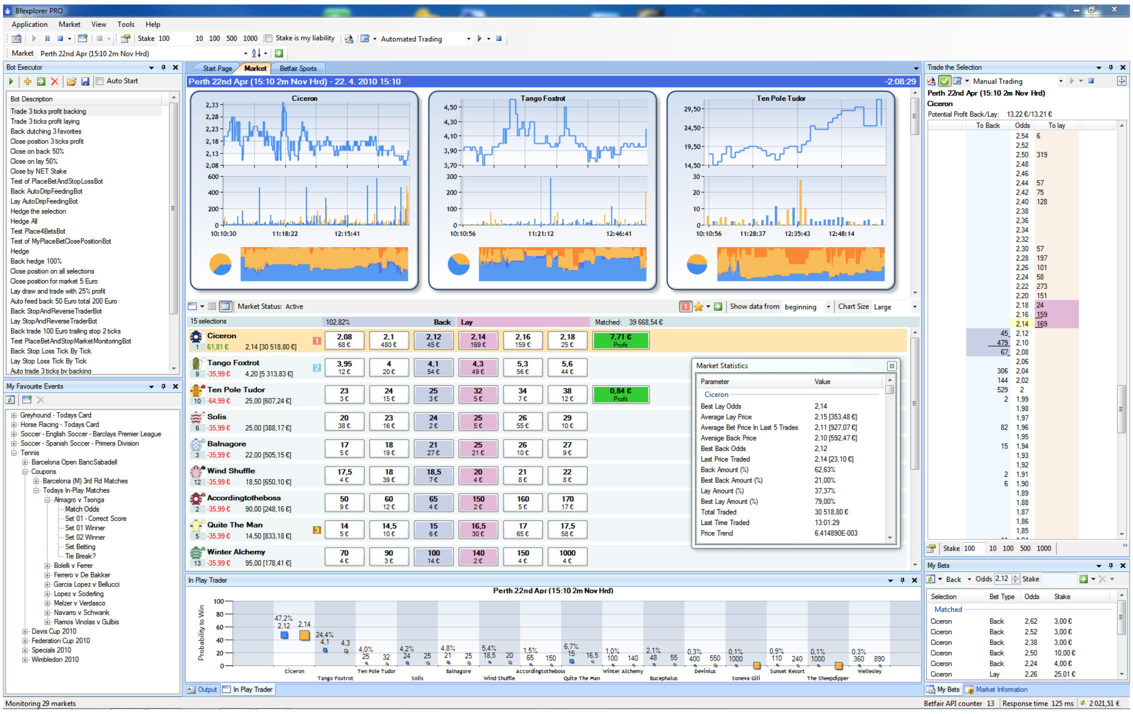
Before calibrating ensure the Match type, the Market Odds type, the Current Score and the Server are set correctly

Match type: 3 set match (tie-break allowed in final set)

Market Odds type: ● Winning Match ○ Winning Set

Player	Server	Sets	Games	Points	Tie-break Points	Predicted Odds	Probability of winning Game	Probability of winning Set	Probability of winning Match
Current Score									
Player A	●	0	4	0	0	1.19	85.6%	98.3%	84.4%
Player B		0	1	0	0	6.39	14.4%	1.7%	15.6%
☑ **Player B Wins Game**									
Player A		0	4	0	0	1.23	20.0%	91.6%	81.2%
Player B	●	0	2	0	0	5.31	80.0%	8.4%	18.8%
☑ **Player B Wins Set**									
Player A	●	0	0	0	0	2.65	85.6%	61.0%	37.8%
Player B		1	0	0	0	1.61	14.4%	39.0%	62.2%
☑ **Custom Score 1**									
Player A	●	0	0	0	0	1.5	85.6%	61.0%	66.7%
Player B		0	0	0	0	3	14.4%	39.0%	33.3%
□ **Player A Wins Set**									
Player A	●	1	0	0	0	1.17	85.6%	61.0%	85.2%
Player B		0	0	0	0	6.74	14.4%	39.0%	14.8%
□ **Custom Score 2**									
Player A	●	0	0	0	0	1.5	85.6%	61.0%	66.7%
Player B		0	0	0	0	3	14.4%	39.0%	33.3%

Figure 2.9: Tennis trader interface of Bet Angel [Ang14]



Bfexplorer PRO
Application Market View Tools Help
Stake 100 10 100 500 1000 Stake is my liability Automated Trading
Start Page Market Betfair Sports
Ciceron
Tango Foxtrot
Ten Pole Tudor
Show data from beginning Chart Size Large
In Play Trader
Perth 22nd Apr (15:10 2m Nov Hrd)
Probability to Win
Trade the Selection
Manual Trading
To Back Odds To lay
Monitoring 29 markets

Market: Perth 22nd Apr (15:10 2m Nov Hrd)

Perth 22nd Apr (15:10 2m Nov Hrd) - 22. 4. 2010 15:10 -2:08:29

Bot Executor

□ Auto Start

Bot Description

- Trade 3 ticks profit backing
- Trade 3 ticks profit laying
- Back dutching 3 favorites
- Close position 3 ticks profit
- Close on back 50%
- Close on lay 50%
- Close by NET Stake
- Test of PlaceBetAndStopLossBot
- Back AutoDripFeedingBot
- Lay AutoDripFeedingBot
- Hedge the selection
- Hedge All
- Test Place4BetsBot
- Test of MyPlaceBetClosePositionBot
- Hedge
- Back hedge 100%
- Close position on all selections
- Close position for market 5 Euro
- Lay draw and trade with 25% profit
- Auto feed back 50 Euro total 200 Euro
- Back StopAndReverseTraderBot
- Lay StopAndReverseTraderBot
- Back trade 100 Euro trailing stop 2 ticks
- Test PlaceBetAndStopMarketMonitoringBot
- Back Stop Loss Tick By Tick
- Lay Stop Loss Tick By Tick
- Auto trade 3 ticks by backing

My Favourite Events

- Greyhound - Todays Card
- Horse Racing - Todays Card
- Soccer - English Soccer - Barclays Premier League
- Soccer - Spanish Soccer - Primera Division
- Tennis
  - Barcelona Open BancSabadell
  - Coupons
    - Barcelona (M) 3rd Rd Matches
    - Todays In-Play Matches
      - Almagro v Tsonga
        - Match Odds
        - Set 01 - Correct Score
        - Set 01 Winner
        - Set 02 Winner
        - Set Betting
        - Tie Break?
    - Bolelli v Ferrer
    - Ferrero v De Bakker
    - Garcia Lopez v Bellucci
    - Lopez v Soderling
    - Melzer v Verdasco
    - Navarro v Schwank
    - Ramos Vinolas v Gulbis
- Davis Cup 2010
- Federation Cup 2010
- Specials 2010
- Wimbledon 2010

Market Status: Active

15 selections		102,82%		Back	Lay			Matched: 39 668,54 €
Ciceron 1 61,81 € 2,14 [30 518,80 €]	1	2,08 68 €	2,1 480 €	2,12 45 €	2,14 169 €	2,16 159 €	2,18 25 €	7,71 € Profit
Tango Foxtrot 9 -35,99 € 4,20 [5 313,83 €]	2	3,95 12 €	4 20 €	4,1 54 €	4,3 49 €	5,3 56 €	5,6 44 €	
Ten Pole Tudor 10 -64,99 € 25,00 [607,24 €]		23 3 €	24 15 €	25 3 €	32 5 €	34 7 €	38 12 €	0,84 € Profit
Solis 6 -35,99 € 25,00 [388,17 €]		20 38 €	23 16 €	24 2 €	25 5 €	26 55 €	29 10 €	
Balnagore 3 -35,99 € 22,00 [505,15 €]		17 5 €	18 19 €	21 27 €	25 21 €	26 10 €	27 9 €	
Wind Shuffle 12 -35,99 € 18,50 [650,10 €]		17,5 4 €	18 39 €	18,5 7 €	20 4 €	21 8 €	22 8 €	
Accordingtotheboss 2 -35,99 € 90,00 [248,16 €]		50 9 €	60 12 €	65 4 €	150 2 €	160 5 €	170 17 €	
Quite The Man 5 -35,99 € 14,50 [833,18 €]	3	14 5 €	14,5 10 €	15 6 €	16,5 30 €	17 65 €	17,5 58 €	
Winter Alchemy 13 -35,99 € 95,00 [178,41 €]		70 4 €	90 3 €	100 14 €	140 2 €	150 4 €	1000 4 €	

Market Statistics

Parameter	Value
Ciceron	
Best Lay Odds	2,14
Average Lay Price	2,15 [353,48 €]
Average Bet Price In Last 5 Trades	2,11 [927,07 €]
Average Back Price	2,10 [592,47 €]
Best Back Odds	2,12
Last Price Traded	2,14 [23,10 €]
Back Amount (%)	62,63%
Best Back Amount (%)	21,00%
Lay Amount (%)	37,37%
Best Lay Amount (%)	79,00%
Total Traded	30 518,80 €
Last Time Traded	13:01:29
Price Trend	6,414890E-003

Perth 22nd Apr (15:10 2m Nov Hrd)

Ciceron

Potential Profit Back/Lay: 13,22 €/13,21 €

Stake 100 10 100 500 1000

My Bets

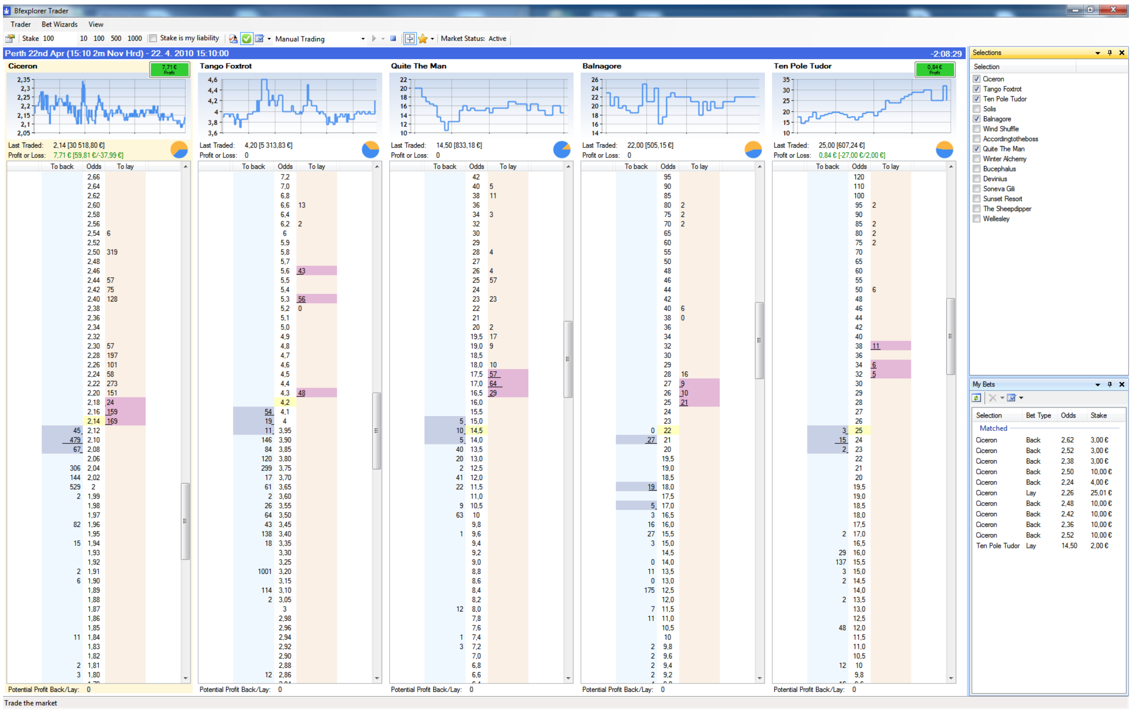
Back Odds 2.12 Stake

Selection	Bet Type	Odds	Stake
Matched			
Ciceron	Back	2,62	3,00 €
Ciceron	Back	2,52	3,00 €
Ciceron	Back	2,38	3,00 €
Ciceron	Back	2,50	10,00 €
Ciceron	Back	2,24	4,00 €
Ciceron	Lay	2,26	25,01 €

My Bets | Market Information

Betfair API counter 13 | Response time 125 ms | 2 021,51 €

Figure 2.10: Guided user interface of BFExplorer, simple betting [Bel14a]



Bfexplorer Trader

Stake 100 ☐ Stake is my liability · Manual Trading · Market Status: Active

Perth 22nd Apr (15:10 2m Nov Hrd) - 22. 4. 2010 15:10:00 · -2:08:29

Ciceron · 7,71 € Profit

Last Traded: 2,14 [30 518,80 €]
Profit or Loss: 7,71 € [59,81 €/-37,99 €]

To back	Odds	To lay
	2,66	
	2,64	
	2,62	
	2,60	
	2,58	
	2,56	
	2,54	6
	2,52	
	2,50	319
	2,48	
	2,46	
	2,44	57
	2,42	75
	2,40	128
	2,38	
	2,36	
	2,34	
	2,32	
	2,30	57
	2,28	197
	2,26	101
	2,24	58
	2,22	273
	2,20	151
	2,18	24
	2,16	159
	2,14	169
45	2,12	
479	2,10	
67	2,08	
	2,06	
306	2,04	
144	2,02	
529	2	
2	1,99	
	1,98	
	1,97	
82	1,96	
	1,95	
15	1,94	
	1,93	
	1,92	
2	1,91	
6	1,90	
	1,89	
	1,88	
	1,87	
	1,86	
	1,85	
11	1,84	
	1,83	
	1,82	
2	1,81	
3	1,80	

Potential Profit Back/Lay: 0

Tango Foxtrot

Last Traded: 4,20 [5 313,83 €]
Profit or Loss: 0

To back	Odds	To lay
	7,2	
	7,0	
	6,8	
	6,6	13
	6,4	
	6,2	2
	6	
	5,9	
	5,8	
	5,7	
	5,6	43
	5,5	
	5,4	
	5,3	56
	5,2	0
	5,1	
	5,0	
	4,9	
	4,8	
	4,7	
	4,6	
	4,5	
	4,4	
	4,3	48
	4,2	
54	4,1	
19	4	
11	3,95	
146	3,90	
84	3,85	
120	3,80	
299	3,75	
17	3,70	
61	3,65	
2	3,60	
26	3,55	
64	3,50	
43	3,45	
138	3,40	
18	3,35	
	3,30	
	3,25	
1001	3,20	
	3,15	
114	3,10	
2	3,05	
	3	
	2,98	
	2,96	
	2,94	
	2,92	
	2,90	
	2,88	
12	2,86	

Potential Profit Back/Lay: 0

Quite The Man

Last Traded: 14,50 [833,18 €]
Profit or Loss: 0

To back	Odds	To lay
	42	
	40	5
	38	11
	36	
	34	3
	32	
	30	
	29	
	28	4
	27	
	26	4
	25	57
	24	
	23	23
	22	
	21	
	20	2
	19,5	17
	19,0	9
	18,5	
	18,0	10
	17,5	57
	17,0	64
	16,5	29
	16,0	
	15,5	
5	15,0	
10	14,5	
5	14,0	
40	13,5	
20	13,0	
2	12,5	
41	12,0	
22	11,5	
	11,0	
9	10,5	
63	10	
	9,8	
1	9,6	
	9,4	
	9,2	
	9,0	
	8,8	
	8,6	
	8,4	
	8,2	
12	8,0	
	7,8	
	7,6	
1	7,4	
3	7,2	
	7,0	
	6,8	
	6,6	

Potential Profit Back/Lay: 0

Balnagore

Last Traded: 22,00 [505,15 €]
Profit or Loss: 0

To back	Odds	To lay
	95	
	90	
	85	
	80	2
	75	2
	70	2
	65	
	60	
	55	
	50	
	48	
	46	
	44	
	42	
	40	6
	38	0
	36	
	34	
	32	
	30	
	29	
	28	16
	27	9
	26	10
	25	21
	24	
	23	
0	22	
27	21	
	20	
	19,5	
	19,0	
	18,5	
19	18,0	
	17,5	
5	17,0	
3	16,5	
16	16,0	
27	15,5	
3	15,0	
	14,5	
0	14,0	
11	13,5	
0	13,0	
175	12,5	
	12,0	
7	11,5	
11	11,0	
	10,5	
	10	
2	9,8	
2	9,6	
2	9,4	
2	9,2	

Potential Profit Back/Lay: 0

Ten Pole Tudor · 0,84 € Profit

Last Traded: 25,00 [607,24 €]
Profit or Loss: 0,84 € [-27,00 €/2,00 €]

To back	Odds	To lay
	120	
	110	
	100	
	95	2
	90	
	85	2
	80	2
	75	2
	70	
	65	
	60	
	55	
	50	6
	48	
	46	
	44	
	42	
	40	
	38	11
	36	
	34	6
	32	5
	30	
	29	
	28	
	27	
	26	
3	25	
15	24	
2	23	
	22	
	21	
	20	
	19,5	
	19,0	
	18,5	
	18,0	
	17,5	
2	17,0	
	16,5	
29	16,0	
137	15,5	
3	15,0	
2	14,5	
	14,0	
2	13,5	
	13,0	
	12,5	
48	12,0	
	11,5	
	11,0	
	10,5	
12	10	
	9,8	

Potential Profit Back/Lay: 0

Selections

Selection
☑ Ciceron
☑ Tango Foxtrot
☑ Ten Pole Tudor
☐ Solis
☑ Balnagore
☐ Wind Shuffle
☐ Accordingtotheboss
☑ Quite The Man
☐ Winter Alchemy
☐ Bucephalus
☐ Devinius
☐ Soneva Gili
☐ Sunset Resort
☐ The Sheepdipper
☐ Wellesley

My Bets

Selection	Bet Type	Odds	Stake
Matched			
Ciceron	Back	2,62	3,00 €
Ciceron	Back	2,52	3,00 €
Ciceron	Back	2,38	3,00 €
Ciceron	Back	2,50	10,00 €
Ciceron	Back	2,24	4,00 €
Ciceron	Lay	2,26	25,01 €
Ciceron	Back	2,48	10,00 €
Ciceron	Back	2,42	10,00 €
Ciceron	Back	2,36	10,00 €
Ciceron	Back	2,52	10,00 €
Ten Pole Tudor	Lay	14,50	2,00 €

Trade the market

Figure 2.11: Ladder interface of BFExplorer, multiple selections [Bel14a]

should happen when it is triggered, independently of the market on wich it is used. The trading mechanisms already developed are the following:

- Open
- Close
- Swing
- Dutching
- Trailing-stop
- Scalping

The working principles of each one of these trading mechanisms are detailed at 2.1. The innovation present at this piece of software is that it creates a high-level interface in Java for users to develop their own agents, without having to handle and know Betfair API from scratch. JBet also includes a simulation mode to test agents.

At the moment there is not a user friendly GUI available, manily because this library is intended to be used by programmers that are comfortable with Java language, altough there is a ladder with read-only purposes to maintain track of trades created by agents. An example of this ladder is shown in figure B.1. For this thesis, JBet was extended with a new trading mechanism (Dealer of section 2.1.5) using the existent interfaces.

2.2.6 Comparison

	Programming Interface	Expansion Capability (Flexibility)	Replay	Simulation Mode	Documentation
Market Feeder Pro	Proprietary scripting language	Medium	True	Weak	Good
Gruss Bet	Excel	Medium	False	False	Medium
Bet Angel	Excel/Triggers	Weak	False	Weak	Good
BF Explorer	Visual Studio	Good	False	Weak	Good
JBet	Java	Good	True	Good	Weak

Table 2.1: Comparison between trading softwares

Table 2.1 presents the most important features to be compared among the trading tools described in previous section. These features include the existence of a programming interface, expansion capability of each of the softwares compared, replay capacity, simulation mode and documentation.

Concerning the programming interface, each software has it own programming interface. The most robust are JBet and BFExplorer because offer the possibility to use a real programming language with a lot of documentation, Java by JBet and Visual Basic and C# by BFExplorer. MarketFeeder Pro offers a proprietary scripting language and Gruss Bet and Bet Angel only offers the possibility to create triggers using Excel spresdsheets.

Only BFExplorer and JBet are truly flexible allowing anyone to develop it's own trading agents with the advantage for JBet that the most important trading mechanisms are already developed and ready to be used, like scalping, dutching, swing among others, explained in section 2.1.

Regarding replay capacity, the clear winner is JBet as it allows to have limitless replay allowing this way to test strategies with unlimited past data for better future predictions. MarketFeeder Pro also have a replay functionality but is somewhat limited. It is called TimeWarp and only allows to replay with a limited ladder leaving some action on markets outside (Betfair log files).

Only GrussBet has no simulation mode, but only JBet does this simulation the way it is supposed to be. Other softwares simulation modes do not account for market liquidity. This means that if someone is using simulation mode at, for example BF Explorer, and place a back bet with 500€ it will be matched even if only 10€ are available on market to match that back bet. So, simulation on this softwares are for purely test plain strategies with low value stakes. On the opposite side, JBet does this the right way. Using the 500€ back bet example, only 10€ will be matched and the remaining 490€ will continue waiting to be matched. This way, it is not only possible to test plain strategies, but also the evolution of that strategies with increasing stakes and assess if it a short or long term strategy.

Documentation is good in all softwares but JBet. It is a product in development and documentation needs more work because can be a little difficult to start programming using JBet and not have a way to know how it works. All other softwares have a reasonably good documentation.

2.3 Machine Learning Packages

This thesis uses machine learning techniques in order to optimize the results of the trading agents developed using the JBet interface described in 2.2.5.

There are many machine learning software available in the market and the well known are probably RapidMiner created by the company with the same name [Rap14], WEKA (Waikato Environment for Knowledge Analysis) developed by Waikato University in New Zeland [Uni14b] and Encog made by Heaton Research [Res13].

Due to the documentation availability, ease of use with a very intuitive interface for model creation, model quantity to be used and export of these models for later integration in JBet code already developed by its creators (Department of Informatics Engineering in School of Engineering, Porto University) the choice for this thesis was the RapidMiner software. Anyway, every model developed during this work using RapidMiner can also be created in any other machine learning software with an adaptation of the exported model for integration with JBet.

In this work, it is only consedered supervising learning, wich is a kind of learning that is done through the analysis of historical data. The historical data includes a set of examples (by now called ExampleSet). Each example contains a situation or input (described by a set of attributes) and the associated result or output (the class to be learned).

Next paragraphs describe some of the existing supervising learning models included in the Rapid Miner software, suitable to be used in the optimization of the results of the trading agents considered in this work. It is important to mention that the description of the classifiers was taken from [AH12].

Decision Tree

Generates a Decision Tree for classification of both nominal and numerical data.

A decision tree is a tree-like graph or model. It is more like an inverted tree because it has its root at the top and it grows downwards. This representation of the data has the advantage compared with other approaches of being meaningful and easy to interpret. The goal is to create a classification model that predicts the value of a target attribute (often called class or label) based on several input attributes of the ExampleSet. In RapidMiner an attribute with label role is predicted by the Decision Tree operator. Each interior node of tree corresponds to one of the input attributes. The number of edges of a nominal interior node is equal to the number of possible values of the corresponding input attribute. Outgoing edges of numerical attributes are labeled with disjoint ranges. Each leaf node represents a value of the label attribute given the values of the input attributes represented by the path from the root to the leaf.

Decision Trees are generated by recursive partitioning. Recursive partitioning means repeatedly splitting on the values of attributes. In every recursion the algorithm follows the following steps:

- An attribute A is selected to split on. Making a good choice of attributes to split on each stage is crucial to generation of a useful tree. The attribute is chosen depending upon a selection criterion usually based on the statistically significance of the attribute.
- Examples in the ExampleSet are sorted into subsets, one for each value of the attribute A in case of a nominal attribute. In case of numerical attributes, subsets are formed for disjoint ranges of attribute values.
- A tree is returned with one edge or branch for each subset. Each branch has a descendant subtree or a label value produced by applying the same algorithm recursively.

In general, the recursion stops when all the examples or instances have the same label value, i.e. the subset is pure. Or recursion may stop if most of the examples are of the same label value. This is a generalization of the first approach; with some error threshold. However other halting conditions can be considered, such as:

- There are less than a certain number of instances or examples in the current subtree. This can be adjusted by using the minimal size for split parameter.
- No attribute reaches a certain threshold. This can be adjusted by using the minimum gain parameter.
- The maximal depth is reached. This can be adjusted by using the maximal depth parameter.

Pruning is a technique in which leaf nodes that do not add nothing significantly meaningful to the discriminative power of the decision tree are removed. This is done to convert an over-specific or over-fitted tree to a more general form in order to enhance its predictive power on unseen datasets. Pre-pruning is a type of pruning performed parallel to the tree creation process. Post-pruning, on the other hand, is done after the tree creation process is complete.

Random Forest

This operator generates a set of a specified number of random trees i.e. called a random forest. The resulting model is a voting model of all the trees.

The representation of the data in form of a tree has the advantage compared with other approaches of being meaningful and easy to interpret. Each interior node of the tree corresponds to one of the input attributes. The number of edges of a nominal interior node is equal to the number of possible values of the corresponding input attribute. Outgoing edges of numerical attributes are labeled with disjoint ranges. Each leaf node represents a value of the label attribute given the values of the input attributes represented by the path from the root to the leaf.

Rule Induction

This operator learns a pruned set of rules with respect to the information gain from a given set of examples (ExampleSet). The Rule Induction operator works similar to the propositional rule learner named 'Repeated Incremental Pruning to Produce Error Reduction' [Coh95]. Starting with the less prevalent classes, the algorithm iteratively grows and prunes rules until no positive examples are left or the error rate is greater than 50%.

Rule Set learners are often compared to Decision Tree learners. Rule Sets have the advantage to be easier to understand, representable in first order logic (easy to implement in languages like Prolog) and prior knowledge can be added easily. The major disadvantages of Rule Sets were that they scaled poorly with training set size and had problems with noisy data. The RIPPER algorithm (which this operator implements) pretty much overcomes these disadvantages. The major problem with Decision Trees is overfitting i.e. the model works very well on the training set but does not perform well on the validation set. Reduced Error Pruning (REP) is a technique that tries to overcome overfitting. After various improvements and enhancements over the period of time REP changed to IREP, IREP* and RIPPER.

This operator uses two phases: the growing phase and the prunning phase. In the growing phase, for each rule greedily conditions are added to the rule until it is perfect (i.e. 100% accurate). The procedure tries every possible value of each attribute and selects the condition with highest information gain. In the prune phase, for each rule any final sequences of the antecedents is pruned with the pruning metric p/(p+n).

k-Nearest Neighbor

This operator generates a k-Nearest Neighbor model from the input ExampleSet. This model can be a classification or regression model depending on the input ExampleSet.

The k-Nearest Neighbor algorithm is based on learning by analogy, that is, by comparing a given test example with training examples that are similar to it. The training examples are described by n attributes. Each example represents a point in an n-dimensional space. In this way, all of the training examples are stored in an n-dimensional pattern space. When an unknown example is given, a k-nearest neighbor algorithm searches the pattern space for the k training examples that are closest to the unknown example. These k training examples are the k "nearest neighbors" of the unknown example. "Closeness" is defined in terms of a distance metric, such as the Euclidean distance.

The k-nearest neighbor algorithm is amongst the simplest of all machine learning algorithms: an example is classified by a majority vote of its neighbors, with the example being assigned to the class most common amongst its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the example is simply assigned to the class of its nearest neighbor.The same method can be used for regression, by simply assigning the label value for the example to be the average of the values of its k nearest neighbors. It can be useful to weight the contributions of the neighbors, so that the nearer neighbors contribute more to the average than the more distant ones.

The neighbors are taken from a set of examples for which the correct classification (or, in the case of regression, the value of the label) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

The basic k-Nearest Neighbor algorithm is composed of two steps: Find the k training examples that are closest to the unseen example and take the most commonly occurring classification for these k examples (or, in the case of regression, take the average of these k label values).

Naive Bayes

This operator generates a Naive Bayes classification model.

A Naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem (from Bayesian statistics) with strong (naive) independence assumptions. A more descriptive term for the underlying probability model would be 'independent feature model'. In simple terms, a Naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class (described by an attribute) is unrelated to the presence (or absence) of any other feature. For example, a fruit may be considered to be an apple if it is red, round, and about 4 inches in diameter. Even if these features depend on each other or upon the existence of the other features, a Naive Bayes classifier considers all of these properties to independently contribute to the probability that this fruit is an apple.

The advantage of the Naive Bayes classifier is that it only requires a small amount of training data to estimate the means and variances of the variables necessary for classification. Because independent variables are assumed, only the variances of the variables for each label need to be determined and not the entire covariance matrix.

Neural Net

This operator learns a model by means of a feed-forward neural network trained by a back propagation algorithm (multi-layer perceptron). This operator cannot handle polynominal attributes.

The coming paragraphs explain the basic ideas about neural networks, need-forward neural networks, back-propagation and multi-layer perceptron.

An artificial neural network (ANN), usually called neural network (NN), is a mathematical model or computational model that is inspired by the structure and functional aspects of biological neural networks. A neural network consists of an interconnected group of artificial neurons, and it processes information using a connectionist approach to computation (the central connectionist principle is that mental phenomena can be described by interconnected networks of simple and often uniform units). In most cases an ANN is an adaptive system that changes its structure based on external or internal information that flows through the network during the learning phase. Modern neural networks are usually used to model complex relationships between inputs and outputs or to find patterns in data.

A feed-forward neural network is an artificial neural network where connections between the units do not form a directed cycle. In this network, the information moves in only one direction, forward, from the input nodes, through the hidden nodes (if any) to the output nodes. There are no cycles or loops in the network.

Back propagation algorithm is a supervised learning method which can be divided into two phases: propagation and weight update. The two phases are repeated until the performance of the network is good enough. In back propagation algorithms, the output values are compared with the correct answer to compute the value of some predefined error-function. By various techniques, the error is then fed back through the network. Using this information, the algorithm adjusts the weights of each connection in order to reduce the value of the error function by some small amount. After repeating this process for a sufficiently large number of training cycles, the network will usually converge to some state where the error of the calculations is small. In this case, one would say that the network has learned a certain target function.

A multilayer perceptron (MLP) is a feed-forward artificial neural network model that maps sets of input data onto a set of appropriate output. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. MLP utilizes back propagation for training the network. This class of networks consists of multiple layers of computational units, usually interconnected in a feed-forward way. In many applications the units of these networks apply a sigmoid function as an activation function. In this operator usual sigmoid function is used as the activation function. Therefore, the values ranges of the attributes should be scaled to -1 and +1. This can be done through the normalize parameter. The type of the output node is sigmoid if the learning data describes a classification task and linear if the learning data describes a numerical regression task.

Linear Regression

This operator calculates a linear regression model from the input ExampleSet.

Regression is a technique used for numerical prediction. Regression is a statistical measure that attempts to determine the strength of the relationship between one dependent variable (class) and a series of other changing variables known as independent variables (regular attributes). Just like Classification is used for predicting categorical labels, Regression is used for predicting a continuous value. For example, we may wish to predict the salary of university graduates with 5 years of work experience, or the potential sales of a new product given its price. Regression is often used to determine how much specific factors such as the price of a commodity, interest rates, particular industries or sectors influence the price movement of an asset.

Linear regression attempts to model the relationship between a scalar variable and one or more explanatory variables by fitting a linear equation to observed data. For example, one might want to relate the weights of individuals to their heights using a linear regression model.

This operator calculates a linear regression model. It uses the Akaike criterion for model selection. The Akaike information criterion is a measure of the relative goodness of a fit of a statistical model. It is grounded in the concept of information entropy, in effect offering a relative measure of the information lost when a given model is used to describe reality. It can be said to describe the tradeoff between bias and variance in model construction, or loosely speaking between accuracy and complexity of the model.

	Input Types	**Output Types**	**Normalization**	**Human Comprehension**
Rule Induction	Nominal/Numerical	Nominal	Not Needed	Yes
k-NN	Nominal/Numerical	Nominal/Numerical	Not Needed	No
Naive Bayes	Nominal/Numerical	Nominal/Numerical	Not Needed	No
Decision Tree	Nominal/Numerical	Nominal	Not Needed	Yes
Random Forest	Nominal/Numerical	Nominal/Numerical	Not Needed	No
Neural Net	Numerical	Numerical	Needed	No
Linear Regression	Numerical	Numerical	Needed	No

Table 2.2: Comparison between machine learning models

Table 2.2 summarizes the main differences between the models explained in this section. The parameters compared are important on the context of the development of our trading agents using the framework described in this thesis because the data will be transfered from an agent to the model and then the model will be exported to integration in the same agent. For this reason, is important to compare the Input Types, Output Types and Normalization. Human Comprehension is important to eventually do a manual check for possible errors in analysis.

Chapter 3

Framework

In this chapter will be described a framework used to create trading agents that uses plain strategies combined with machine learning. Machine learning is included to improve the base plain strategy. Athough it is important to keep in mind that if a plain strategy has clearly negative results, the machine learning stages defined on the framework will not help turning that negative results into profit. Following the steps described is possible to have a consistent and secure methodology to develop this type of agents.

3.1 Description

Figure 3.1 schematically represents the base framework for all the agents developed during this thesis. At start, a plain strategy (see definition at glossary) is tested in simulation mode, to verify its viability. A clear negative plain stategy is not helpful. For most of the traders it will not be always clear when some strategy has a positive or negative tendency mainly because it is extremelly complicated to make evaluations in the long run. As any trader knows, this evaluations are essential to assess how a strategy behaves. Eventually, after testing the strategy in simulation, some parameters of it could be subject to changes to better adapt to market changes. This empirical refining of parameters is important to find out the best parameter adjustment to mimize loss or increase profit in the plain strategy defining stage. If needed, the cycle of testing and refining is restarted until the results of plain strategy are acceptable.

The next phase is the reuse or creation of new data from original market data that is suitable to train and test the model to be created. This data could be raw data from markets like time of the day, number of selections, amongst others, or could be new data calculated using raw data, that better suits the model testing. For this framework, this data is the output of the agent running with only the plain strategy. This data (a table with examples for modelation) is then divided in two parts, one for training the model and other to test it.

After this division is complete, the next step is to create the model with any of the machine learning packages described in section2.3.

The model is now created, so is time to first train it and then test it, using the previously created data. Possibly, the results will allow for some improvement after a first iteration, so, as done with the plain strategy, there is a need to refine model parameters, also called hyper parameters. This empirical refinement of hyper parameters will, in theory, improve model responses and the cycle of testing and refinement could be done until the model gives the best possible results.

It is now time to integrate the model trained and tested into the plain strategy. Before every major decision point, the agent should use the model to ask if the plain strategy should be applied or not at that moment in the market. In sport events markets, usually this decision points are the entry or not on respective market. But in theory, this decision points could be anything that the trader wants, be it at wich time close the position on market or even the stake to be used depending on market values.

After testing again the plain strategy filtered with the model created on machine learning stage, the results should be better than it was without the model integration. Be it the loss reduction or profit increase, the goal is achieved. It is now time for the developer to decide if he wants to go into real mode, with real money, or restart the whole cycle again with a new plain strategy. The most common and obvious decision is that if the whole product has profit, it should be passed to real mode, with the reserve that on betting exchanges nothing is guaranteed and all strategies could possibly be obsolete at some point in time.

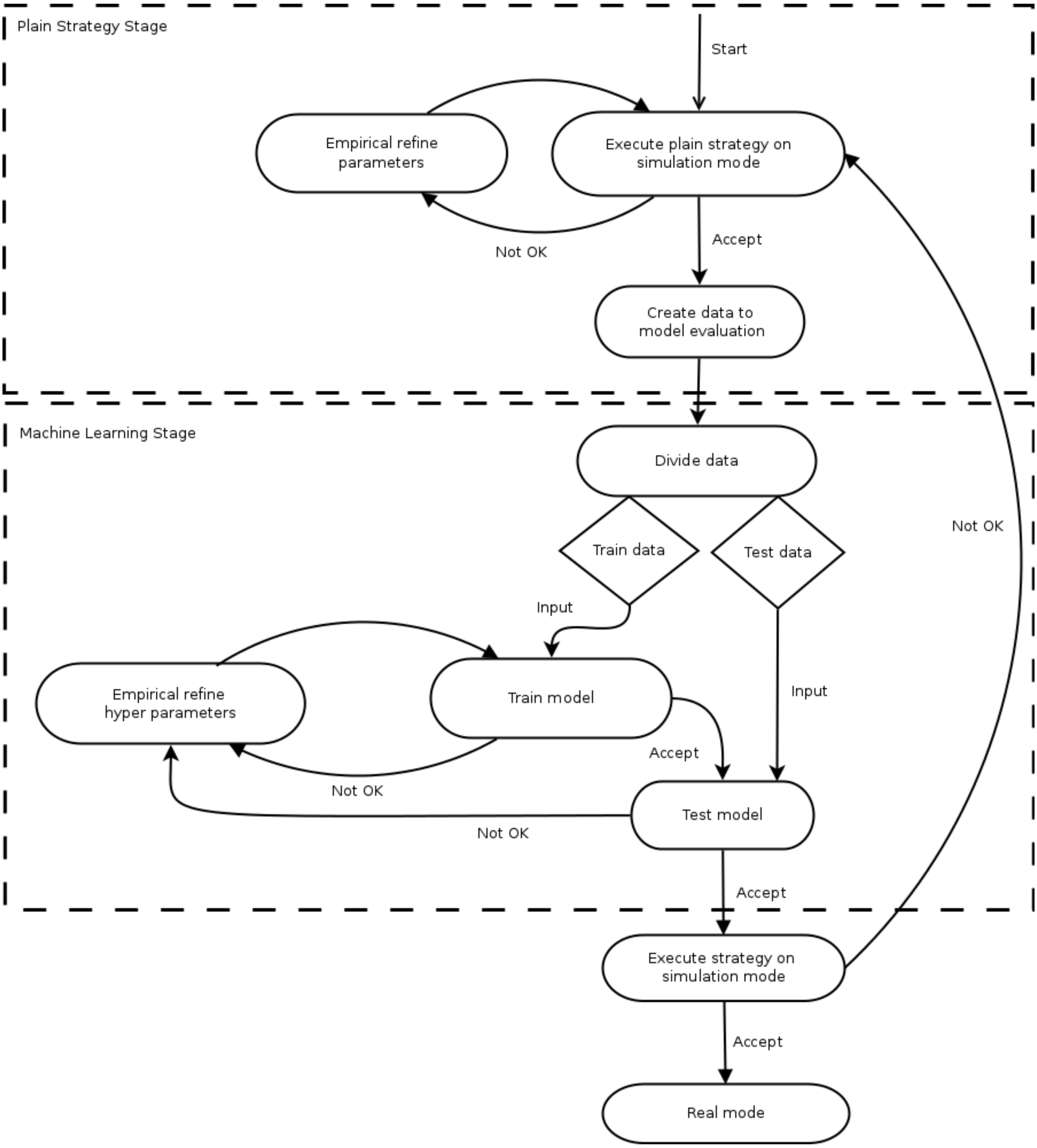

Plain Strategy Stage
Start
Empirical refine parameters
Execute plain strategy on simulation mode
Not OK
Accept
Create data to model evaluation
Machine Learning Stage
Divide data
Train data
Test data
Not OK
Input
Input
Empirical refine hyper parameters
Train model
Accept
Not OK
Test model
Not OK
Accept
Execute strategy on simulation mode
Accept
Real mode

Figure 3.1: Schematic representation of framework

Chapter 4

Implementation

In this secion will be explained how two trading agents that follows the framework presented in section 3 were implemented using JBet. Also, to create them, was necessary to develop a new trading mechanism that extend JBet functionalities, whose implementation will also be described.

4.1 Dealer Agent

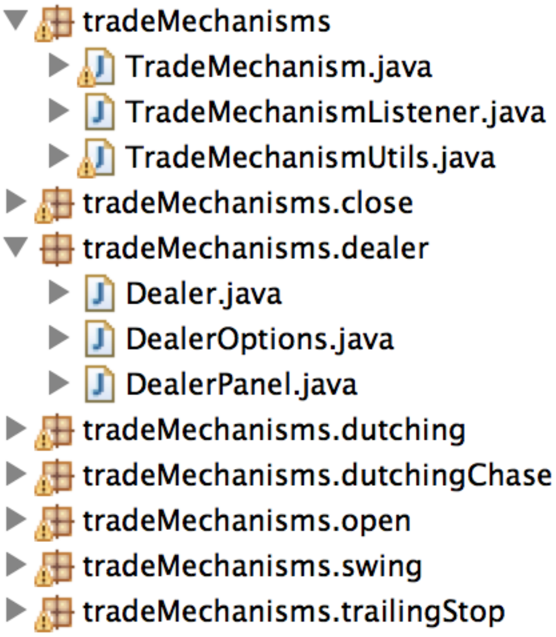
According to Oxford Dictionary, a dealer is "a person who buys and sells goods" [Uni14a]. The first and most common synonym for "dealer" is trader. So in this context, we can assume that every trader that work on Betfair and other beting exchanges are dealers. This is true, regardless of the nature of the trader, wether it be a person or an automated trading mechanism. From the market perspective, everyone is an entity that open and close positions (buy and sell) and its nature is not relevant.

4.1.1 The Plain Strategy Stage

The implemented agent follows a plain strategy for pre-live horse racing. It tries to take advantage of the market fluctuations (around an *axisOdd*) before each race start, placing a back and a lay simultaneously, and try to close with profit by matching both in a time frame. If it is not successful, a stoploss will be triggered and will try to close with loss at best price available.

When agent is running, it will enter on horse races with more than 4 horses and at least one of that horses has the odd above 6.0 and lower than 12.0. These are horses that are not favourites to win the race, but at the same time have a suitable liquidity to be traded on. When remaining time to the start of the race is less than 5 minutes, the agent will create a dealer for each one of the horses filtered before.

To create a dealer agent with JBet, that followed the principles described in section 2.1.5, it was necessary to create a trading mechanism inside JBet that followed those rules. This allows that, from now on, every JBet agent can use a dealer by simply invocate that trading mechanism not being dependent of the plain strategy and markets where it is used used, wether it be for horses, football, car racings, etc.



```
▼ tradeMechanisms
    ▶ TradeMechanism.java
    ▶ TradeMechanismListener.java
    ▶ TradeMechanismUtils.java
▶ tradeMechanisms.close
▼ tradeMechanisms.dealer
    ▶ Dealer.java
    ▶ DealerOptions.java
    ▶ DealerPanel.java
▶ tradeMechanisms.dutching
▶ tradeMechanisms.dutchingChase
▶ tradeMechanisms.open
▶ tradeMechanisms.swing
▶ tradeMechanisms.trailingStop
```

Figure 4.1: Class structure of dealer trading mechanism

In Appendix A, figure A.1 shows the DealerAgent GUI with statistics fields resulting from the execution of one trading mechanism dealer.

4.1.2 Trading Mechanism Dealer

The trading mechanism dealer is a set of predifined actions that, when invocated, will make a back and a lay bet simultaneously in different odds. This trading mechanism will then wait that both bets are matched to return a profit. In the case that market has a upward or downward tendency (making it impossible to match both bets), dealer will then trigger a stoploss condition to avoid biggers losses.

As any other trading mechinism in JBet, dealer has 3 classes where everything is implemented. Figure 4.1 shows the class structure of the dealer trading mechanism and names of the three classes created.

DealerOptions.java

This class implements all the parameters needed to invoke a dealer.

DealerPanel.java

This class implements a graphical user interface to be invoked by agents that use the trading mechanism dealer. It is used JSwing to create JLabel's and JComboBox'es allowing users to easily define DealerOptions.java parameters. If the implemented agent defines itself DealerOptions.java paramenters, it is not necessary to invoke DealerPanel.java.

Dealer.java

This class implements all the dealer logic, creating necessary open and closing positions. This class extends the superclass *TradeMechanism.java* and manages all the listeners subscriptions that implement the *TradeMechanismListener.java* interface, following an event based architecture for the Dealer process (or mechanism).

The parameters available to define the *DealerOptions.java* are shown in listing 4.1 and explained in detail in the next paragraphs..

```java
private double stake;
private double axisOdd;
private int ticksBackProfit;
private int ticksLayProfit;
private int ticksLossBack;
private int ticksLossLay;
private int backOrLayStake;
private double percentageOpen;
private boolean insistOpen;
private boolean useIpKeep;
private int waitFramesOpen;
private int waitFramesBestPrice;
private boolean forceCloseOnStopLoss;
private boolean useStopProfifInBestPrice;
private boolean goOnfrontInBestPrice;
private int delayIgnoreStopLoss;
private int waitFramesLay1000;
```

Listing 4.1: DealerOptions.java parameters available

The trading mechanism dealer will use the values of each of these parameters to act on the market chosen by the agent and it is mandatory that all parameters have a value assigned for correct execution of the dealer. To prevent failures, all parameters have a default value. These default values represent the most common choice, although it is recommended to modify them to ensure that dealer will execute according to the will of the trader.

Stake is the amount of money that dealer will use to open a position on market. As it will open two positions at the same time, the stake value will be used to open the position defined at *backOrLayStake* that can be a *BetData.back* or a *BetData.lay*. If the agent defines that the stake is to be used with a back, the lay stake will be calculated depending on the odds chosen (see equation 1.4). The odds to open positions are calculated using *ticksBackProfit*, *ticksLayProfit* and *axisOdd* parameters. Usually, *axisOdd* is the odd available at that moment to bet on and it is always updating to reflect market immediatly available odd. Using this odd, dealer will then calculate the odds to open positions using *ticksBackProfit* and *ticksLayProfit*. As an example, if *axisOdd* is 1.68 and *ticksBackProfit* and *ticksLayProfit* have a value of 1, dealer will open two positions at the same time backing at 1.69 and laying at 1.67. This means that if the trade is successful and both postions are matched, the agent will have 2 ticks of profit.

As a security measure, this trading mechaninsm also implements a stoploss functionality. *ticksLossBack* and *ticksLossLay* represent the ticks to wait until trigger the stoploss. Using the previous example, and assuming that both *ticksLossBack* and *ticksLossLay* have a value of 2, after backing and laying at 1.69 and 1.67 respectively, dealer will maintain the odds 1.71 and 1.65 in memory. If the market is in uptrend and reaches 1.71 before matching the lay bet at 1.67 it will trigger a stoploss cancelling the lay bet at 1.67 and opening a new lay, with recalculated stake, at market best price, returning in a loss for the agent. This works also if the market is in dowtrend and its mechanism is similar, but dealer will cancel the unmatched back position and will open a new one at best price.

The *percentageOpen* parameter, by default 1.0, represents the required percentage to close the trade after the initial bet is matched. For example, if *percentageOpen* is set to 0.5, when the agent starts a trade and half of the stake is matched, it will move on to the next phase of the dealer.

The *insistOpen* parameter is used if trader wants the agent to insist open a position even if the market is suspended. When the market reactivates, the agent will continue trying to open the previous unmatched position. The *useIPKeep* parameter is similar to the previous one, and if set to true, the agent will keep the bets even if the market turns in-play. By default, when a race starts (turn in-play), all the bets are cancelled.

The *waitFramesOpen* and *waitFramesBestPrice* are the parameters that control the time to wait to match positions. When a trade is started, the two positions waiting to be matched will wait until the value *waitFramesOpen* be reached. This value represents the number of frames (updates of prices from the market, for this agent, 2 FPS) passed since the trade was started. When in need to close at best price, and in the same way as before, that position will wait the number of frames represented by *waitFramesBestPrice*. After the *waitFramesBestPrice* timeout, the trade mechanism will close where there is money available on the ladder. It is important to mention that when the agent starts a trade and the *waitFramesOpen* time is reached before one of the bets are matched, both are cancelled and it will not enter the market.

The parameter *forceCloseOnStopLoss* is used when a stoploss condition is reached and in that case, agent will not try to close at best price and will close where there is money available to close the position. This parameter allows for faster closings in case of loss but with a penalty of at least one tick on the ladder since it does not use *waitFramesBestPrice* time.

The parameter *useStopProfitInBestPrice* is useful for volatile markets. If the agent is in state of closing at best price and this parameter is set to true, it will always close that position on the original price requested. Even if the price changes to a better one when trying to close at best price, the agent will maintain the original one. This allows faster closings shortening the exposure time on market with the risk of loosing potential profit if the market is in favor of the agent forecasting. *goOnFrontInBestPrice* is also responsible for shortening the exposure time but in a different way. If the agent is in the state of closing at best price, and *goOnFrontInBestPrice* is set to true, the agent will close one tick below the immediate best offered price and not behind other best offers.

With the parameter *delayIgnoreStopLoss* different than 0 (by default *delayIgnoreStopLoss = 0*), from the moment where a position is opened, the agent will ignore the stoploss functionality for the number of frames defined, with the objective of waiting for market stabilization. If the market has an extremely low liquidity (e.g. market goes in-play and all the bets are cancelled), it is important to ignore stoploss until the market stabilizes (the liquidity fills again).

Finally, when a result becomes impossible, the agent will wait the number of frames defined by *waitFramesLay1000* to try to close in the worst price of the ladder (1000). The probability of closing here is extremely low so after this time the agent assumes the loss.

4.1.3 Machine Learning Stage

Having already the plain strategy phase completed, with all the parameter optimization and refinement possible, is now the moment of apply the Machine Learning process. The start of this phase, as described in figure 3.1, its the creation of data for the model (train and test evaluation). This data is created from the executions on simulation mode of DealerAgent while plain strategy. In the end of each execution of each trading mechanism dealer, it is created one example for the model. The DealerAgent runs many trading mechanisms (one for each filtered horse on each race) that creates a set of examples. The data to be created is represented on listing 4.2.

```
private double matchedAtStartMoment;
private int runners;
private double womDiffAtStartMoment;
private double demandAtStartMoment;
private double percentDiffBack;
private double percentDiffLay;
private boolean profit;
```

Listing 4.2: Fields of the dataset to be created to use in model train and test of DealerAgent

The values of 4.2 were chosen for model creation because they are the ones that better represent the market and it is with them that the model will choose to enter or not - run the dealer trading mechanism or not, for that horse.

The market liquidity, represented by *matchedAtStartMoment*, is the total money traded on the runner. The lower this value get, there will be less market fluctuations. The number of horses in race, represented by *int runners*, is important to evaluate on wich race types there are better results. These two values (*matchedatStartMoment* and *runners*) are directly extracted from market without any kind of post-processing.

On the contrary, all values presented in this paragraph are the result of post-processing some of the atomic market values. The first one to be calculated is the weight of money difference of market - *womDiffAtStartMoment*. To get to this value it is simply calculated the difference of weight of money between the present and a near past represented by 30 frames of market. The line of code where this calculation happens is represented in listing 4.3

```java
ri.womDifAtStartMoment = Utils.getAmountBackFrame(rdAux, 30) -
                         Utils.getAmountLayFrame(rdAux, 0);
```

Listing 4.3: Calculation of weight of money difference

The *demandAtStartMoment* represents the difference of demand between the moment a dealer is started and 30 frames in the past. Theorically, if the demand is greater in present than in the past, the market tendency is a downward one because there are more entities offering money than the ones accepting it. In the case of the demand be smaller in the present than it was in the past, the market has an upward tendency. The ideal range for this value is near zero, point where the market will be stabilized favouring the matching of both bets made at start of a trading mechanism dealer. The calculation of *demandAtStartMoment* is presented in listing 4.4

```java
ri.demandAtStartMoment = UtilsCollectData.getAmountMatchedVariationAVGAxisWindow
                         (rdAux, 0, 30, 3);
```

Listing 4.4: Calculation of demand at start moment

The values *percentDiffBack* and *percentDiffLay* are representative of the existence of a support at immediatly next available odd where the dealer has started. This values represent a percentage in relation to the available amount waiting to be matched at the next odd relative to where dealer opened positions. In the case of the value of *percentDiffBack* be grater than 1.0, there is more money available on the next odd than in the odd where the bet was made. This evaluation is important because it represent a possible pressure point meaning that the prices could possible not pass that odd, avoiding the triggering of stoploss. Obviously, both *percentageDifffBack* and *percentageDiffLay* represents the support on the back and lay bets made by the dealer. Their calculation is made by the code in listing 4.5.

```java
double amountBackN=Utils.getAmountBackOddFrame(rdAux, oddBackN, 0);
double amountLayN=Utils.getAmountLayOddFrame(rdAux, oddLayN, 0);

double amountBack=Utils.getAmountBackOddFrame(rdAux, oddBack, 0);
double amountLay=Utils.getAmountLayOddFrame(rdAux, oddLay, 0);

double percentLay=(amountLay/amountLayN)*100.;
double percentBack=(amountBack/amountBackN)*100.;
```

Listing 4.5: *percentageDiffBack* and *percentageDiffLay* calculation

The last value will be the target/label/output field for the machine learning model and it represents the success or insuccess of a trading mechanism dealer execution. It is a boolean that represents if there was profit on that dealer execution. In the case of the final profit, in euros, is equal or greater than zero, the boolean profit will assume the value true.

The listing 4.6 represents an example of a *.csv* file generated by DealerAgent. Each line represents one execution of a trading mechanism dealer. In one race could be executed several trading mechanism dealers depending on the number of horses with odds between 6.0 and 12.0, five minutes before the start of the race. However, it is created only one dealer in each eligible horse.

```
28989.22 10 96.38000000000001 -16.419833333333326 148.73555020653174
    11.937690348993959 true
25662.33 10 -105.44 -4.901000000000015 14.099754267118284 26.89237334959133 true
22386.25 10 48.67999999999999 -27.237666666666666 14.335539613966947
    213.80622837370242 true
4724.06 15 -7.669999999999998 0.054333333333333185 13.032015065913372
    6.583419253828235 true
10091.13 15 -52.32 -1.4748333333333326 266.72705314009664 288.51617995264405 false
7024.83 6 31.909999999999997 -3.585999999999999 56.50704225352112
    24.201286107992242 true
7542.65 6 -1.6799999999999926 -2.401333333333333 115.11207060765763
    103.48107285524063 false
10112.96 6 1.6500000000000057 1.5066666666666673 106.53774089935762
    318.1999501371229 false
2974.01 12 -7.08 -0.014166666666667046 102.56410256410258 164.03508771929828 true
5635.24 12 17.029999999999998 -0.4661666666666671 222.23816355810618
    48.68701206529454 false
10283.04 9 -63.48000000000002 -3.9181666666666692 562.1270857913366
    120.09025010801209 false
```

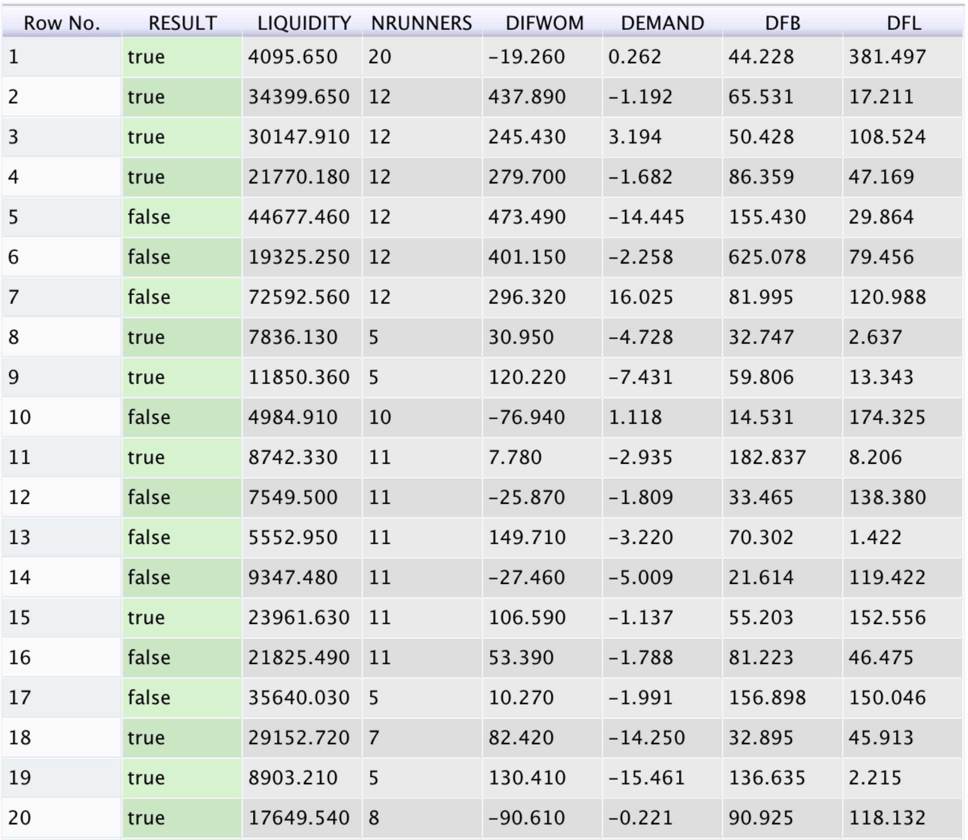
Listing 4.6: File generated by a DealerAgent execution with values to use in train and test of model

After the creation of data is completed, they are divided in two parts: one for training and other for testing and refining the model. In DealerAgent concrete case, the generated data represent 3 months of horse races that are divided in two months for training and one month for testing (January and February of 2014 for training and March of 2014 for testing).

It is then loaded the training data to RapidMiner software. Figure 4.2 represents the loading of a part of the training data.

The learning process of model was created using the scheme shown in figure 4.3. It was created this way, with all datamining classifiers suitable for use (accepting numerical input and nominal output). The classifier used in this example was Rule Induction as we can see in image 4.3.

With the RapidMiner process created and the training data prepared, it is time for training the model. The results are presented in figure 4.4 and are a set of rules that the generated model will use at the time of testing the data. The test process is presented in figure 4.5. The table 4.1 represents the final values, obtained in phase "refinement of hyperparameters" of the framework, that are used to train the machine learning model, that uses a Rule Induction classifier.



Row No.	RESULT	LIQUIDITY	NRUNNERS	DIFWOM	DEMAND	DFB	DFL
1	true	4095.650	20	-19.260	0.262	44.228	381.497
2	true	34399.650	12	437.890	-1.192	65.531	17.211
3	true	30147.910	12	245.430	3.194	50.428	108.524
4	true	21770.180	12	279.700	-1.682	86.359	47.169
5	false	44677.460	12	473.490	-14.445	155.430	29.864
6	false	19325.250	12	401.150	-2.258	625.078	79.456
7	false	72592.560	12	296.320	16.025	81.995	120.988
8	true	7836.130	5	30.950	-4.728	32.747	2.637
9	true	11850.360	5	120.220	-7.431	59.806	13.343
10	false	4984.910	10	-76.940	1.118	14.531	174.325
11	true	8742.330	11	7.780	-2.935	182.837	8.206
12	false	7549.500	11	-25.870	-1.809	33.465	138.380
13	false	5552.950	11	149.710	-3.220	70.302	1.422
14	false	9347.480	11	-27.460	-5.009	21.614	119.422
15	true	23961.630	11	106.590	-1.137	55.203	152.556
16	false	21825.490	11	53.390	-1.788	81.223	46.475
17	false	35640.030	5	10.270	-1.991	156.898	150.046
18	true	29152.720	7	82.420	-14.250	32.895	45.913
19	true	8903.210	5	130.410	-15.461	136.635	2.215
20	true	17649.540	8	-90.610	-0.221	90.925	118.132

Figure 4.2: Portion of the data to train the machine learning model in RapidMiner

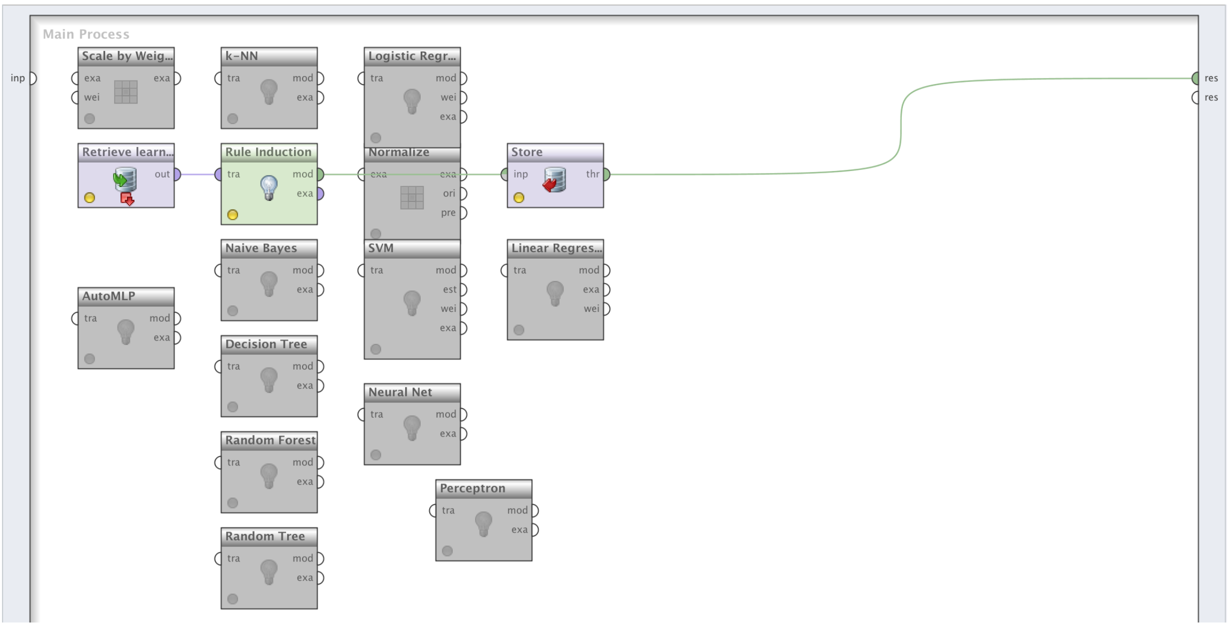
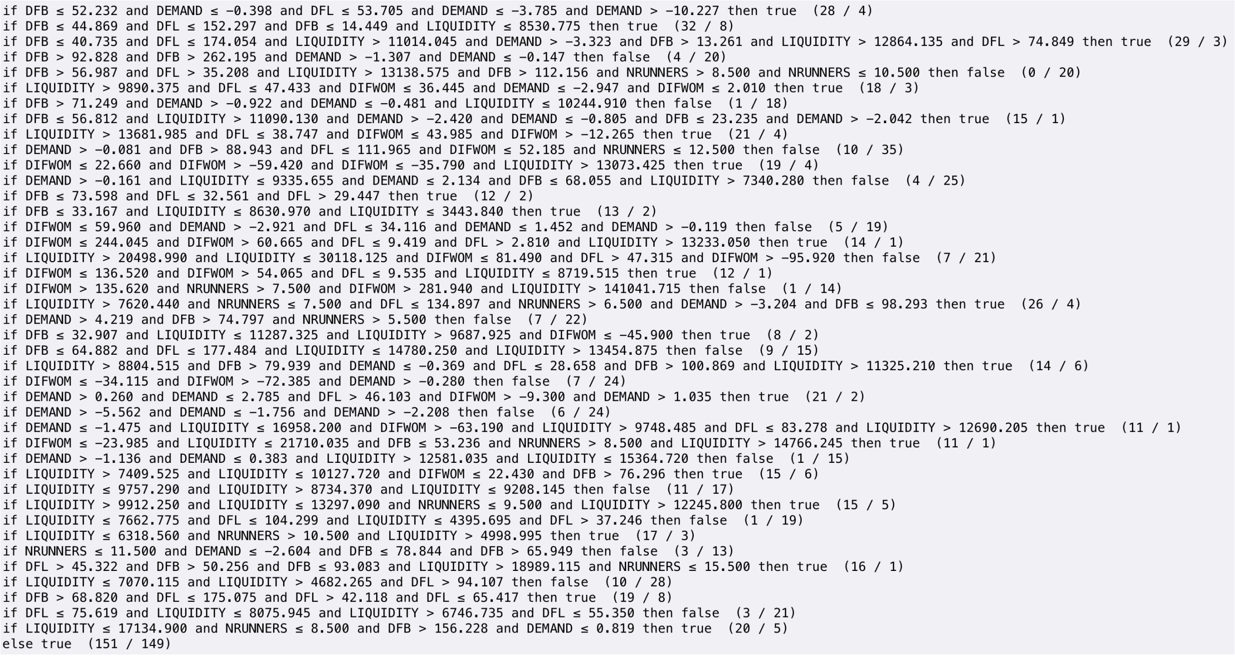


Figure 4.3: Learn process created in RapidMiner



```
if DFB ≤ 52.232 and DEMAND ≤ −0.398 and DFL ≤ 53.705 and DEMAND ≤ −3.785 and DEMAND > −10.227 then true  (28 / 4)
if DFB ≤ 44.869 and DFL ≤ 152.297 and DFB ≤ 14.449 and LIQUIDITY ≤ 8530.775 then true  (32 / 8)
if DFB ≤ 40.735 and DFL ≤ 174.054 and LIQUIDITY > 11014.045 and DEMAND > −3.323 and DFB > 13.261 and LIQUIDITY > 12864.135 and DFL > 74.849 then true  (29 / 3)
if DFB > 92.828 and DFB > 262.195 and DEMAND > −1.307 and DEMAND ≤ −0.147 then false  (4 / 20)
if DFB > 56.987 and DFL > 35.208 and LIQUIDITY > 13138.575 and DFB > 112.156 and NRUNNERS > 8.500 and NRUNNERS ≤ 10.500 then false  (0 / 20)
if LIQUIDITY > 9890.375 and DFL ≤ 47.433 and DIFWOM ≤ 36.445 and DEMAND ≤ −2.947 and DIFWOM ≤ 2.010 then true  (18 / 3)
if DFB > 71.249 and DEMAND > −0.922 and DEMAND ≤ −0.481 and LIQUIDITY ≤ 10244.910 then false  (1 / 18)
if DFB ≤ 56.812 and LIQUIDITY > 11090.130 and DEMAND > −2.420 and DEMAND ≤ −0.805 and DFB ≤ 23.235 and DEMAND > −2.042 then true  (15 / 1)
if LIQUIDITY > 13681.985 and DFL ≤ 38.747 and DIFWOM ≤ 43.985 and DIFWOM > −12.265 then true  (21 / 4)
if DEMAND > −0.081 and DFB > 88.943 and DFL ≤ 111.965 and DIFWOM ≤ 52.185 and NRUNNERS ≤ 12.500 then false  (10 / 35)
if DIFWOM ≤ 22.660 and DIFWOM > −59.420 and DIFWOM ≤ −35.790 and LIQUIDITY > 13073.425 then true  (19 / 4)
if DEMAND > −0.161 and LIQUIDITY ≤ 9335.655 and DEMAND ≤ 2.134 and DFB ≤ 68.055 and LIQUIDITY > 7340.280 then false  (4 / 25)
if DFB ≤ 73.598 and DFL ≤ 32.561 and DFL > 29.447 then true  (12 / 2)
if DFB ≤ 33.167 and LIQUIDITY ≤ 8630.970 and LIQUIDITY ≤ 3443.840 then true  (13 / 2)
if DIFWOM ≤ 59.960 and DEMAND > −2.921 and DFL ≤ 34.116 and DEMAND ≤ 1.452 and DEMAND > −0.119 then false  (5 / 19)
if DIFWOM ≤ 244.045 and DIFWOM > 60.665 and DFL ≤ 9.419 and DFL > 2.810 and LIQUIDITY > 13233.050 then true  (14 / 1)
if LIQUIDITY > 20498.990 and LIQUIDITY ≤ 30118.125 and DIFWOM ≤ 81.490 and DFL > 47.315 and DIFWOM > −95.920 then false  (7 / 21)
if DIFWOM ≤ 136.520 and DIFWOM > 54.065 and DFL ≤ 9.535 and LIQUIDITY ≤ 8719.515 then true  (12 / 1)
if DIFWOM > 135.620 and NRUNNERS > 7.500 and DIFWOM > 281.940 and LIQUIDITY > 141041.715 then false  (1 / 14)
if LIQUIDITY > 7620.440 and NRUNNERS ≤ 7.500 and DFL ≤ 134.897 and NRUNNERS > 6.500 and DEMAND > −3.204 and DFB ≤ 98.293 then true  (26 / 4)
if DEMAND > 4.219 and DFB > 74.797 and NRUNNERS > 5.500 then false  (7 / 22)
if DFB ≤ 32.907 and LIQUIDITY ≤ 11287.325 and LIQUIDITY > 9687.925 and DIFWOM ≤ −45.900 then true  (8 / 2)
if DFB ≤ 64.882 and DFL ≤ 177.484 and LIQUIDITY ≤ 14780.250 and LIQUIDITY > 13454.875 then false  (9 / 15)
if LIQUIDITY > 8804.515 and DFB > 79.939 and DEMAND ≤ −0.369 and DFL ≤ 28.658 and DFB > 100.869 and LIQUIDITY > 11325.210 then true  (14 / 6)
if DIFWOM ≤ −34.115 and DIFWOM > −72.385 and DEMAND > −0.280 then false  (7 / 24)
if DEMAND > 0.260 and DEMAND ≤ 2.785 and DFL > 46.103 and DIFWOM > −9.300 and DEMAND > 1.035 then true  (21 / 2)
if DEMAND > −5.562 and DEMAND ≤ −1.756 and DEMAND > −2.208 then false  (6 / 24)
if DEMAND ≤ −1.475 and LIQUIDITY ≤ 16958.200 and DIFWOM > −63.190 and LIQUIDITY > 9748.485 and DFL ≤ 83.278 and LIQUIDITY > 12690.205 then true  (11 / 1)
if DIFWOM ≤ −23.985 and LIQUIDITY ≤ 21710.035 and DFB ≤ 53.236 and NRUNNERS > 8.500 and LIQUIDITY > 14766.245 then true  (11 / 1)
if DEMAND > −1.136 and DEMAND ≤ 0.383 and LIQUIDITY > 12581.035 and LIQUIDITY ≤ 15364.720 then false  (1 / 15)
if LIQUIDITY > 7409.525 and LIQUIDITY ≤ 10127.720 and DIFWOM ≤ 22.430 and DFB > 76.296 then true  (15 / 6)
if LIQUIDITY ≤ 9757.290 and LIQUIDITY > 8734.370 and LIQUIDITY ≤ 9208.145 then false  (11 / 17)
if LIQUIDITY > 9912.250 and LIQUIDITY ≤ 13297.090 and NRUNNERS ≤ 9.500 and LIQUIDITY > 12245.800 then true  (15 / 5)
if LIQUIDITY ≤ 7662.775 and DFL ≤ 104.299 and LIQUIDITY ≤ 4395.695 and DFL > 37.246 then false  (1 / 19)
if LIQUIDITY ≤ 6318.560 and NRUNNERS > 10.500 and LIQUIDITY > 4998.995 then true  (17 / 3)
if NRUNNERS ≤ 11.500 and DEMAND ≤ −2.604 and DFB ≤ 78.844 and DFB > 65.949 then false  (3 / 13)
if DFL > 45.322 and DFB > 50.256 and DFB ≤ 93.083 and LIQUIDITY > 18989.115 and NRUNNERS ≤ 15.500 then true  (16 / 1)
if LIQUIDITY ≤ 7070.115 and LIQUIDITY > 4682.265 and DFL > 94.107 then false  (10 / 28)
if DFB > 68.820 and DFL ≤ 175.075 and DFL > 42.118 and DFL ≤ 65.417 then true  (19 / 8)
if DFL ≤ 75.619 and LIQUIDITY ≤ 8075.945 and LIQUIDITY > 6746.735 and DFL ≤ 55.350 then false  (3 / 21)
if LIQUIDITY ≤ 17134.900 and NRUNNERS ≤ 8.500 and DFB > 156.228 and DEMAND ≤ 0.819 then true  (20 / 5)
else true  (151 / 149)
```

Figure 4.4: Rules generated as result of the learning process of model in RapidMiner

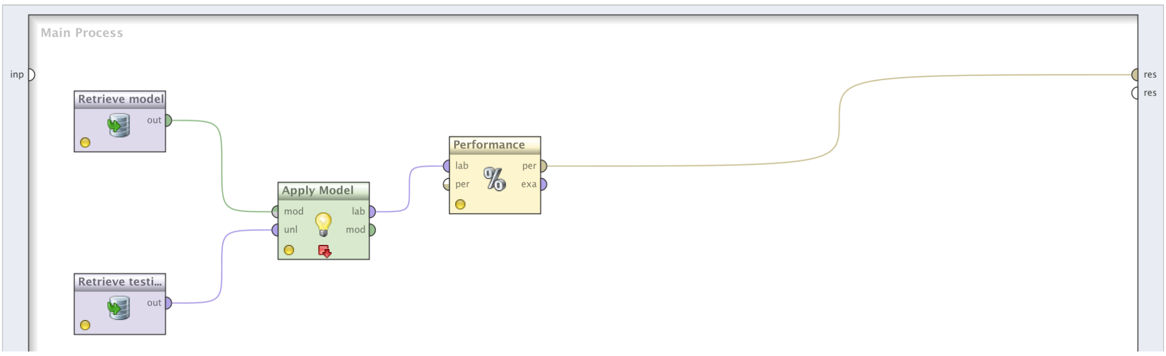


Figure 4.5: Testing process created in RapidMiner

Criterion	Information Gain
Sample Ratio	0.7
Pureness	0.9
Minimal prune benefit	0.1
Local random seed	Yes

Table 4.1: Hyper parameters for Rule Induction classifier for the final refinement of Dealer machine learning model

After execute the test of model, with the previous selected test data (March of 2014), are then presented the performance values. In figure 4.6 is illustrated the performance values for this test, and it is possible to see that the percentage of hits was of 52.33%.

Is then necessary to do an export of the model to a file in order to DealerAgent be able to read it and know when it should or not enter a race according to model. The process of export is represented in figure 4.7

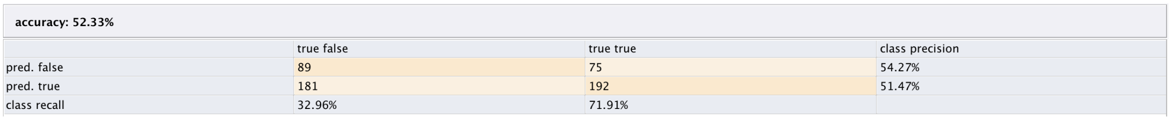
This model is exported to a *.mod* file and in *DealerAgent.java* it is loaded. From this moment on, whenever the DealerAgent finds a suitable race to be traded on according to the parameters of the plain strategy, it will ask the model if it should or not enter being the model responsible for the final decision of trading in a horse or not according to the values presented by the market where the horse is.

4.2 Horse Lay Agent

This is another agent developed in this thesis scope. This agent also follows the structure of the framework presented in 3 and acts on horse races of United Kingdom. It tries to take advantage of the market speculation making a lay bet in non favourite horses, at lower odds than the ones presented (by that horses) at start of the races.

4.2.1 The Plain Strategy Stage

The plain strategy of this agent is of easy understanding: in live horse markets, create a lay at low odds in horses that are not favourites and wait that in any moment of the race the lay is matched by someone else. It is important to note that in a single race several lays can be created, one for each horse that matches the criteria defined by the plain strategy. In the event that several lays are created in the same race, as soon as any of these lays is matched, all of the other lays are immediatly cancelled by the agent. Theorically, if a not favourite horse has his odd at a low value,



accuracy: 52.33%

	true false	true true	class precision
pred. false	89	75	54.27%
pred. true	181	192	51.47%
class recall	32.96%	71.91%	

Figure 4.6: Results of testing the model with pre selected testing data (March 2014) in RapidMiner

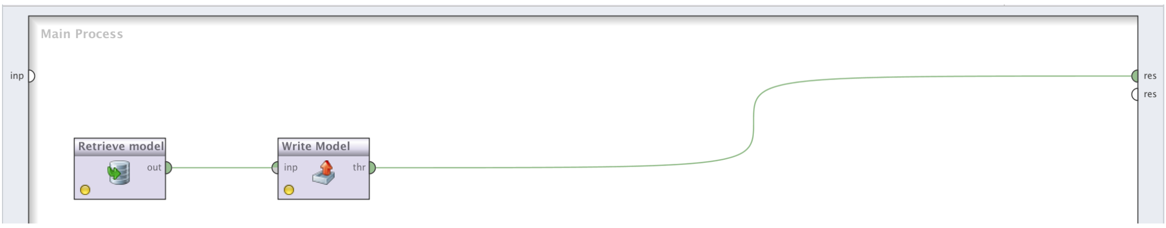


Figure 4.7: Procees of exporting model to external file to be used by DealerAgent

every other horses will see their odds rising and for that reason is is not probable that two lays are matched at the sime time. As soon as a lay is matched, the remaining unmatched lays are cancelled. In summary, the agent will remain with only one lay bet matched in the market, a lay bet to a horse who was not favourite to win.

The markets where the agent works are only United Kingdom races because this is the place where usually there is more liquidity in horse races, meaning a greater speculation essential for the proper functioning of this agent.

The criteria for each agent execution are the following:

- Lay at odd of 2 in horses with initial odd above 3
- Lay at odd of 2 in horses with initial odd above 4
- Lay at odd of 2 in horses with initial odd above 5
- Lay at odd of 2 in horses with initial odd above 6
- Lay at odd of 3 in horses with initial odd above 4
- Lay at odd of 3 in horses with initial odd above 5
- Lay at odd of 3 in horses with initial odd above 6
- Lay at odd of 4 in horses with initial odd above 5
- Lay at odd of 4 in horses with initial odd above 6
- Lay at odd of 5 in horses with initial odd above 6

These are the possible combinations to be tested in a database of horse racing, in simulation mode. From all of the combinations, the ones that have the best results will be selected to advance for the next phase of the framework, the machine learning. This processs of choosing the agents with best results is relative to the "empirical refinement of parameters" in the plain strategy stage of the framework presented in figure 3.1. The results of this executions are presented in section 5.2. From all of the listed combinations, the ones selected were:

- Lay at odd of 3 in horses with initial odd above 6

- Lay at odd of 4 in horses with initial odd above 6

All the other combinations were discarded.

In this agent development there was no need to create a specific trading mechanism, unlike in DealerAgent in 4.1, as the plain strategy is simple. This agent only requires to open a lay position for each eligible horse, functionality obviously already implemented in JBet.

4.2.2 The Machine Learning Stage

The process of machine learning in this agent is similar to the one already explained in 4.1.3 for DealerAgent. The only difference are the values created in the HorseLayAgent execution for later training and testing of the model. This values are:

- Number of horses
- Race liquidity (and not the individual horse liquidity)
- Hour of the day
- Race length
- Profit (output value)

Every one of these values are atomic so they are directly extracted from the market without the need of post-processing. It is important to note that the liquidity extracted is the one from the race and not of the horse where the agent works on.

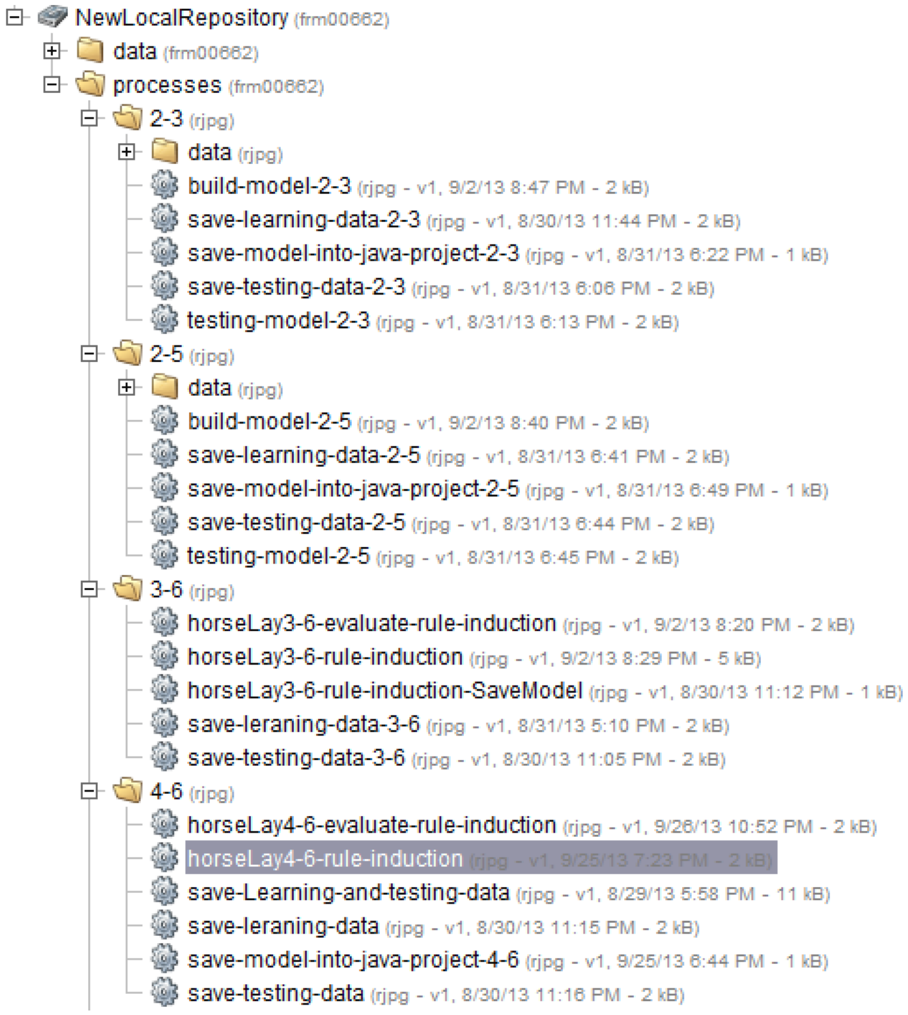
The generated data by the agent is divided in one month for training and two weeks of testing. The data used for training is between December 15th of 2013 and January 14th of 2014 and the testing data is between 15 and 30 of January 2014.

The model, similar to the one presented in section 4.1.3 , is then trained and tested with the data previously created and divided. The model is then exported for integration in *HorseLayAgent.java*.

The chosen classifier was the Rule Induction although the results using Decision Tree were very similar. In figure 4.8 is possible to see the project structure in RapidMiner.

From the "refinement of hyper parameters" phase, described in the framework (figure 3.1), was possible to obtain the best values of each of these parameters for each agent execution. The 4.2 table presents the final values used by the classifier.

The agent will use the model for questioning on wich races it should work on. In the event of a positive response, the agent then selects the eligible horses and executes the plain strategy.



```
NewLocalRepository (frm00662)
  data (frm00662)
  processes (frm00662)
    2-3 (rjpg)
      data (rjpg)
      build-model-2-3 (rjpg - v1, 9/2/13 8:47 PM - 2 kB)
      save-learning-data-2-3 (rjpg - v1, 8/30/13 11:44 PM - 2 kB)
      save-model-into-java-project-2-3 (rjpg - v1, 8/31/13 6:22 PM - 1 kB)
      save-testing-data-2-3 (rjpg - v1, 8/31/13 6:06 PM - 2 kB)
      testing-model-2-3 (rjpg - v1, 8/31/13 6:13 PM - 2 kB)
    2-5 (rjpg)
      data (rjpg)
      build-model-2-5 (rjpg - v1, 9/2/13 8:40 PM - 2 kB)
      save-learning-data-2-5 (rjpg - v1, 8/31/13 6:41 PM - 2 kB)
      save-model-into-java-project-2-5 (rjpg - v1, 8/31/13 6:49 PM - 1 kB)
      save-testing-data-2-5 (rjpg - v1, 8/31/13 6:44 PM - 2 kB)
      testing-model-2-5 (rjpg - v1, 8/31/13 6:45 PM - 2 kB)
    3-6 (rjpg)
      horseLay3-6-evaluate-rule-induction (rjpg - v1, 9/2/13 8:20 PM - 2 kB)
      horseLay3-6-rule-induction (rjpg - v1, 9/2/13 8:29 PM - 5 kB)
      horseLay3-6-rule-induction-SaveModel (rjpg - v1, 8/30/13 11:12 PM - 1 kB)
      save-leraning-data-3-6 (rjpg - v1, 8/31/13 5:10 PM - 2 kB)
      save-testing-data-3-6 (rjpg - v1, 8/30/13 11:05 PM - 2 kB)
    4-6 (rjpg)
      horseLay4-6-evaluate-rule-induction (rjpg - v1, 9/26/13 10:52 PM - 2 kB)
      horseLay4-6-rule-induction (rjpg - v1, 9/25/13 7:23 PM - 2 kB)
      save-Learning-and-testing-data (rjpg - v1, 8/29/13 5:58 PM - 11 kB)
      save-leraning-data (rjpg - v1, 8/30/13 11:15 PM - 2 kB)
      save-model-into-java-project-4-6 (rjpg - v1, 9/25/13 6:44 PM - 1 kB)
      save-testing-data (rjpg - v1, 8/30/13 11:16 PM - 2 kB)
```

Figure 4.8: Project structure of HorseLayAgent model in RapidMiner software

	3-6	4-6
Criterion	Information Gain	Information Gain
Sample Ratio	0.7	0.7
Pureness	0.9	0.9
Minimal prune benefit	0.3	0.1
Local random seed	Not using	True

Table 4.2: Hyper parameters for Rule Induction classifier for the final refinement of HorseLayAgent machine learning model

Chapter 5

Testing and Results

In this section it will be described the results of both developed agents: DealerAgent and HorseLayAgent (4.1 and 4.2). Two types of results are discussed: those that are obtained using only a plain strategy and those obtained with our model. The main objective is to demonstrate the framework (chapter 3) feasibility.

5.1 Dealer Agent

To start, it is important to mention that all graphics presented in this section were created using data from March of 2014.

The first step to test framework feasibility is to run the agent with only the plain strategy (execute plain strategy on simulation mode in figure 3.1). The total profit/loss is presented in function of number of races traded in figure 5.1.

These results were created after several iterations of empirical refinement of the paramenters on the plain strategy and executions on simulation mode, as presented in 3.1. As we can observe, the results were clearly negative. This means that the dealer agent is not efective on the long run having a clear tendency to loose money. Only on month of March of 2014 the loss was about 35€ and the graph shows that the tendecy is to continue loosing money. In figure 5.2 are presented the results for DealerAgent, but this time with integration of the rule induction model. The train set used to train the model were the races that occured between the entire months of January and February of 2014. The first observation to make is that using the model the total profit/loss was slighly better. Even though it is a loss, it was smaller than the loss presented in 5.1 (execution of the agent without the machine learning filter).

There was a positive evolution after applying the machine learning mechanism explained in 3 but DealerAgent does not have a expected positive value in long term [Reb10], even with applying the machine learning filter. In the worst case, it would be expected that the plain strategy could have a expected value near zero in order to have a small increase in profits with the application of the machine learning technique.

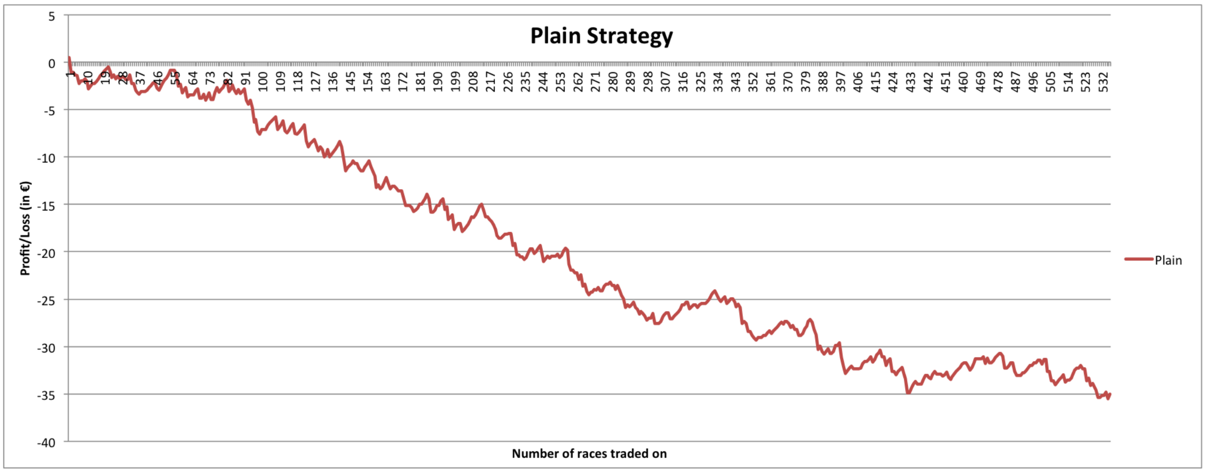


Figure 5.1: Dealer Agent results with plain strategy (without model) for entire month of March of 2014

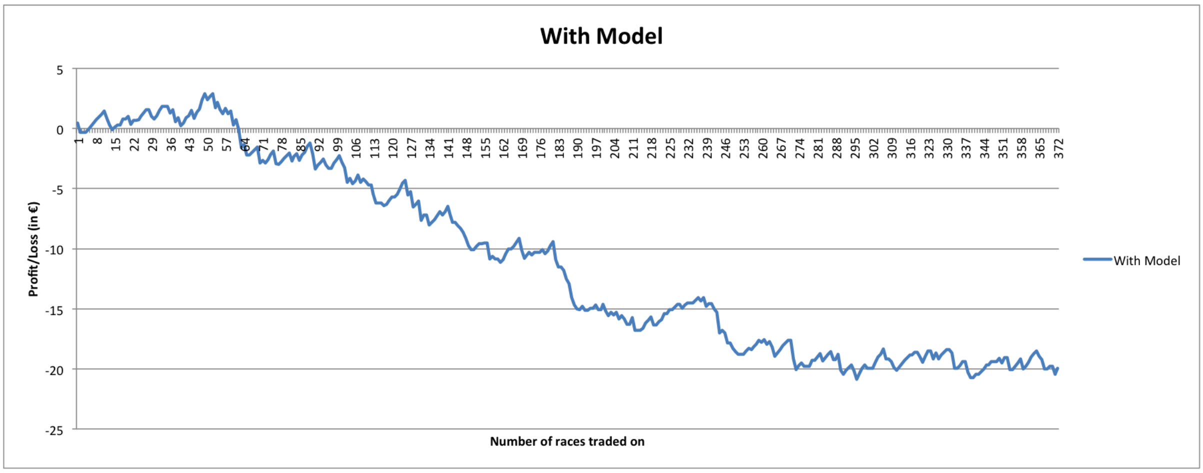


Figure 5.2: Dealer Agent results with rule induction classifier for entire month of March of 2014 (train set: January and February of 2014)

5.2 Horse Lay Agent

In section 4.2 was explained in wich way the empirical refinement of parameters was taking place, executing one agent for each possible odd combination and in the end select those with the best results.

In figure 5.3 is possible to see an example of one combination that was a total failure. The image title is "2-3", wich means that were made lays at odd 2 in horses that in the start of the race had an odd of above 3.

At the final of the test period, is possible to verify that this agent had a loss of about 150€ with stakes of 10€, that represents a considerable amount of loss for the time period tested (between 15 and 30 of January 2014). Furthermore, is possible to observe in the graphic that, although there is a positive peak in the beginning, the long term tendecy will be negative and for that reason is imperative to discard this agent.

In images 5.4 and 5.5 is possible to observe the selected agents according to their results. The red line represents the execution of the agent only with the plain strategy and the blue line represents the same agent but this time with the aplication of the model on top of the plain strategy. The first note to take from images 5.4 and 5.5 is that with model the results improved. This happens because the agent always asks to model if it should enter a race or not and in races that the model think that will not have profit, its answer is negative, avoiding some potential loss.

In both cases (5.4 and 5.5) the horses where the lays were made were clearly not favourites to win that race, with strating odds above 6, but in some moment of the race won advantage over other competitors and the lays made were matched. Is also important to mention that in both cases, the agents with only the plain strategy without the model (red lines), the results in the end of the testing were near zero profit/loss. In image 5.4 is possible to see that the agent with only the plain strategy was much of the time with a positive profit/loss but in the end of the test period suffered a loss resulting in a negative but near zero profit/loss.

In image 5.6 is demonstrated the total profit/loss acheived with the two selected agents (5.4 and 5.5) having a final loss of 25€ in the plain strategy case, but a 95€ of profit when the model was applied to the plain strategy.

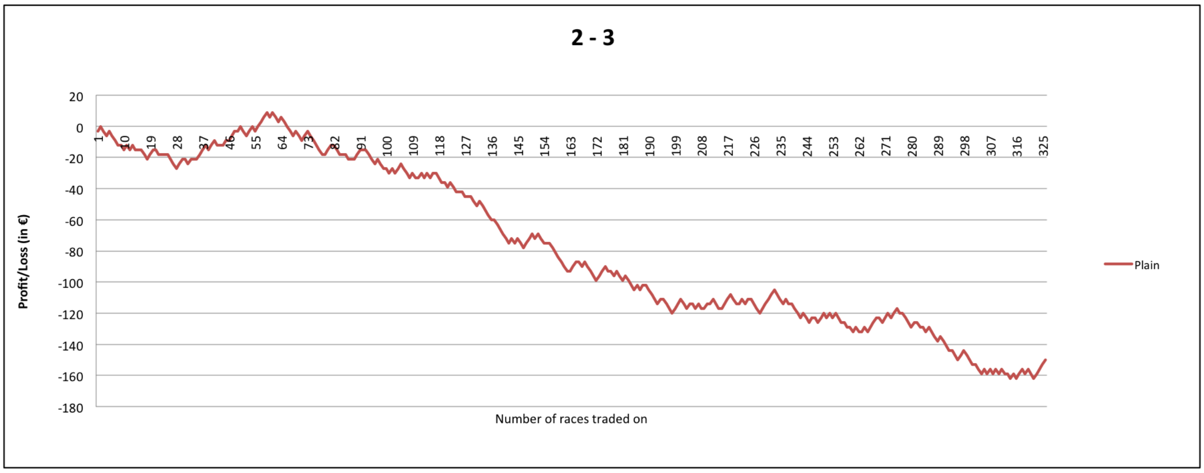


Figure 5.3: Horse Lay Agent results of plain strategy laying at 2 on horses with odd above 3 at race start from from January 15th to January 30th of 2014 (train set: December 15th of 2013 to January 14 of 2014)

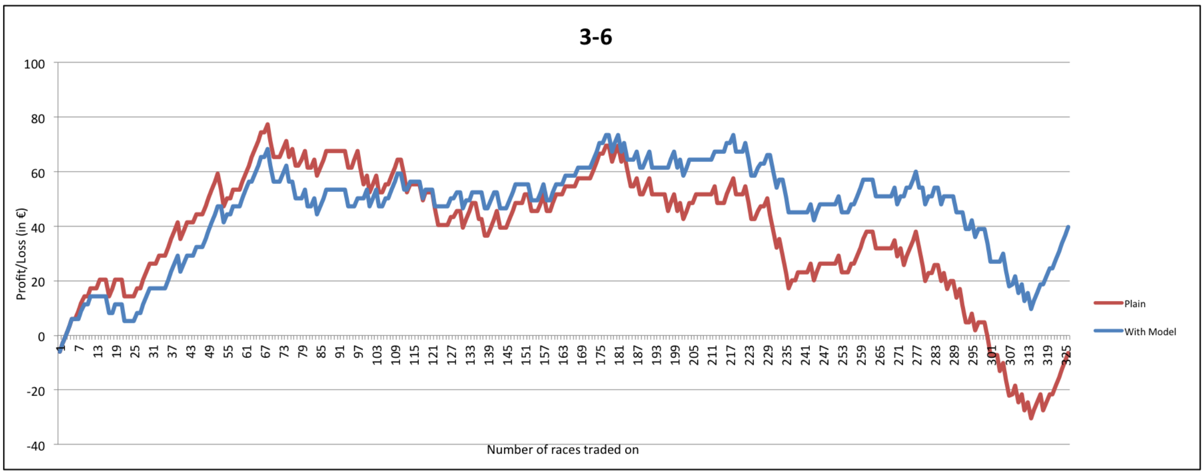


Figure 5.4: Horse Lay Agent results of laying at 3 on horses with odd above 6 at race start with and without rule induction classifier from from January 15th to January 30th of 2014 (train set: December 15th of 2013 to January 14 of 2014)

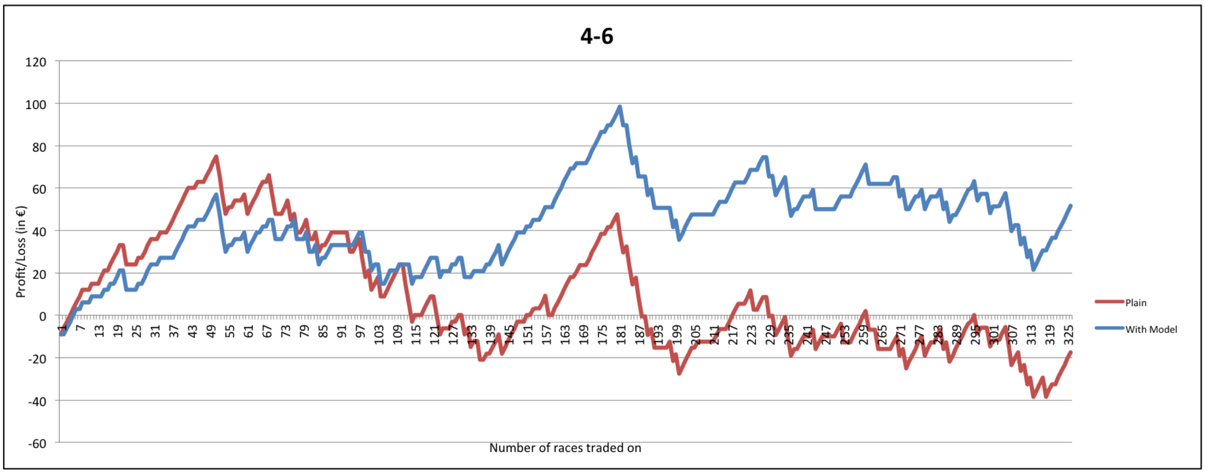


Figure 5.5: Horse Lay Agent results of laying at 4 on horses with odd above 6 at start of the race with and without rule induction classifier from January 15th to January 30th of 2014 (train set: December 15th of 2013 to January 14 of 2014)

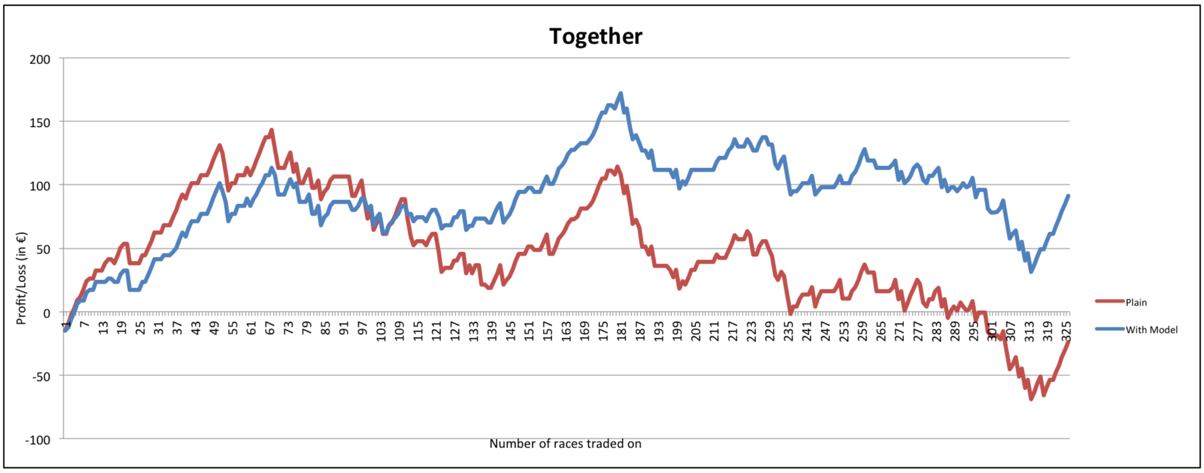


Figure 5.6: Horse Lay Agent combining 5.4 and 5.5

Chapter 6

Conclusions and future work

The first conclusion to take from this work is that a good plain strategy is essencial to have a profitable trading agent in long-term. Without it, no agent will be able to generate profit, the main reason for creating an agent. And as we can see in 5.1 and 5.2, the results with the agents implemented during this work were not the expected ones.

DealerAgent, in 5.1, shows us that even though a plain strategy may sound good in paper, after implementing it the results may be completly different. Even without the machine learning stages described on the framework in 3, the results were negative when it was expected for the profit/loss to be at least near zero.

With HorseLayAgent in 5.2, the results with only the plain strategy were slightly better than in DealerAgent. When the machine learning was implemented for HorseLayAgent the results were even better.

This leads to the second conclusion of this thesis, that after several refinements of the plain strategy, implementing machine learning techniques to the agent can possibly improve the results. The two example agents of this work (DealerAngent and HorseLayAgent) shown that the profit/loss results were better when the machine learning stages defined on the framework were applied.

It is important to say that, even though both agents were developed to work on horse racing markets, JBet and its trading mechanisms allow every Java programmer to create its own agents for any other sports market on Betfair exchange. The most difficult is, with no doubt, finding a good plain strategy. To find one, it is needed a great knowledge of the target markets and how they behave in different conditions.

6.1 Future Work

Regarding the agents developed in this work, a greater time frame for training and testing is needed. DealerAgent possibly has no salvation even if it is trained with more data because the plain strategy behind is clearly a negative one. The same can not be applied to HorseLayAgent. It had a positive result after the machine learning implementation with a train set of only one month.

If this train set is extended, possibly the model will take better decisions when asked by the agent. So, for future work, the HorseLayAgent will be tested in more depth.

Obviously, other work to be done in future is the creation of new agents with JBet. The possibilities to create new agents are virtually infinite. There are many types of markets on Betfair, and everyone has to find the ones where is more confortable with. The next agent to be developed can possibly be intended to use in footabll matches. The reasons for this is that in football matches, specially from England and Spain, there is always a lot of liquidity in circulation.

References

[Ace13] AceOdds. A guide to dutching. Available at http://www.aceodds.com/what-is-dutching.html, 2013. Last accessed: 2014-03-23.

[AH12] Fareed Akthar and Caroline Hahne. *Rapid Miner 5: Operator Reference*, 2012.

[Ang14] Bet Angel. Bet angel - the ultimate betfair toolkit. Available at http://www.betfair.com, 2014. Last accessed: 2014-05-30.

[Bel14a] BeloSoft. Betfair trading software. Available at http://bfexplorer.net, 2014. Last accessed: 2014-05-29.

[Bel14b] BeloSoft. Bfexplorer products. Available at http://bfexplorer.net/Products.aspx, 2014. Last accessed: 2014-05-29.

[Coh95] W.W. Cohen. Fast effective rule induction. In *Proceedings of the Twelfth International Conference on Machine Learning*, pages 115 – 23, San Francisco, CA, USA, 1995.

[DPSW05] Mark Davies, Leyland Pitt, Daniel Shapiro, and Richard Watson. Betfair.com:: Five technology forces revolutionize worldwide wagering. *European Management Journal*, 23(5):533 – 541, 2005.

[FVN10] Egon Franck, Erwin Verbeek, and Stephan Nüesch. Prediction accuracy of different market structures — bookmakers versus a betting exchange. *International Journal of Forecasting*, 26(3):448 – 459, 2010. Sports Forecasting.

[Int14] Ladbrokes International. Betdaq - the people's betting exchange. Available at http://www.betdaq.com, 2014. Last accessed: 2014-06-21.

[Lim13] Betfair Counterparty Services Limited. Betfair cash out. Available at https://www.betfair.com/sport/cashout, 2013. Last accessed: 2014-05-25.

[Lim14a] Betclic Limited. Bet online with betclic. Available at http://en.betclic.com, 2014. Last accessed: 2014-06-21.

[Lim14b] Electraworks Limited. bwin - bet online. Available at http://www.bwin.com, 2014. Last accessed: 2014-06-21.

[Lim14c] Talented Mavericks Limited. Betfair trading software. Available at http://www.geekstoy.com/en, 2014. Last accessed: 2014-06-20.

[Lim14d] The Sporting Exchange Limited. Online betting - sportsbook and exchange at betfair.com. Available at http://www.betfair.com, 2014. Last accessed: 2014-06-21.

[Mak14] Jim Makos. Bet angel: Sports trading software for betfair and betdaq. Available at http://jimmakos.com/bet-angel/, 2014. Last accessed: 2014-06-02.

[Rap14] RapidMiner. Rapidminer studio. Available at http://rapidminer.com/products/rapidminer-studio/, 2014. Last accessed: 2014-06-15.

[Reb10] Paulo Rebelo. Plain strategies with expected positive value. Available at http://www.paulorebelotrader.com/en/2010/10/06/plain-strategies-with-expected-positive-value/, 2010. Last accessed: 2014-05-17.

[Res13] Heaton Research. Encog machine learning framework. Available at http://www.heatonresearch.com/encog, 2013. Last accessed: 2014-06-15.

[RGP13] Ana Paula Rocha Rui Gonçalves and Fernando Lobo Pereira. High Level Architecture for Trading Agents in Betting Exchange Markets. In *Advances in Information Systems and Technologies SE - 46*, volume 206 of *Advances in Intelligent Systems and Computing*, pages 497–510. Springer Berlin Heidelberg, 2013.

[Seb02] Fabrizio Sebastiani. Machine learning in automated text categorization. *ACM Comput. Surv.*, 34(1):1–47, March 2002.

[Sof05a] WellDone Creative Software. Auto-greenup function. Available at http://marketfeeder.co.uk/screenshots/auto-green-up/, 2005. Last accessed: 2014-06-01.

[Sof05b] WellDone Creative Software. Automated betfair trading software - marketfeeder pro - triggered automated betting bot. Available at http://marketfeeder.co.uk, 2005. Last accessed: 2014-05-30.

[Sof10] Gruss Software. Betfair betting assistant. Available at http://www.gruss-software.co.uk, 2010. Last accessed: 2014-04-20.

[Tra12] UK Football Trading. Best betfair trading software - bet angel. Available at http://www.ukfootballtrading.com/bet-angel-betfair-trading-software/, 2012. Last accessed: 2014-05-29.

[Tra14a] Betfair Pro Trader. Weight of money. Available at http://www.betfairprotrader.co.uk/2010/12/weight-of-money.html, 2014. Last accessed: 2014-06-20.

[Tra14b] Traderline. Grid and ladder interfaces. Available at http://www.traderline.org/grid-ladder-interfaces, 2014. Last accessed: 2014-06-20.

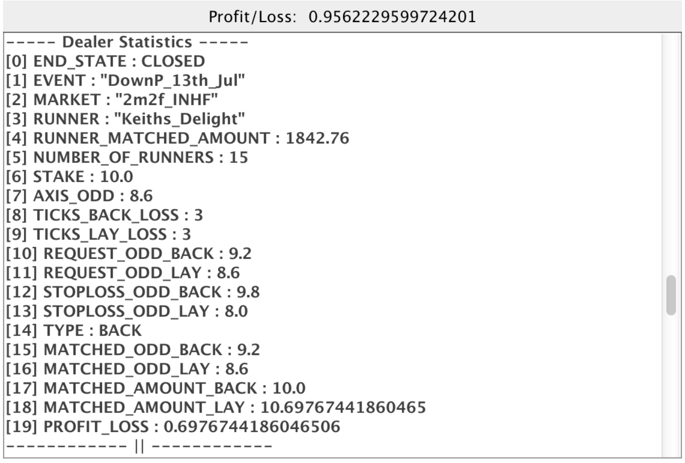
[TZ88] Richard H. Thaler and William T. Ziemba. Anomalies parimutuel betting markets: Racetracks and lotteries. volume 2, pages 161–174. 1988.

[Uni14a] Oxford University. Definition of dealer in english. Available at http://www.oxforddictionaries.com/definition/english/dealer?q=dealer, 2014. Last accessed: 2014-05-01.

[Uni14b] Waikato University. Machine learning group. Available at http://www.cs.waikato.ac.nz/ml/index.html, 2014. Last accessed: 2014-06-15.

Appendix A

Dealer GUI



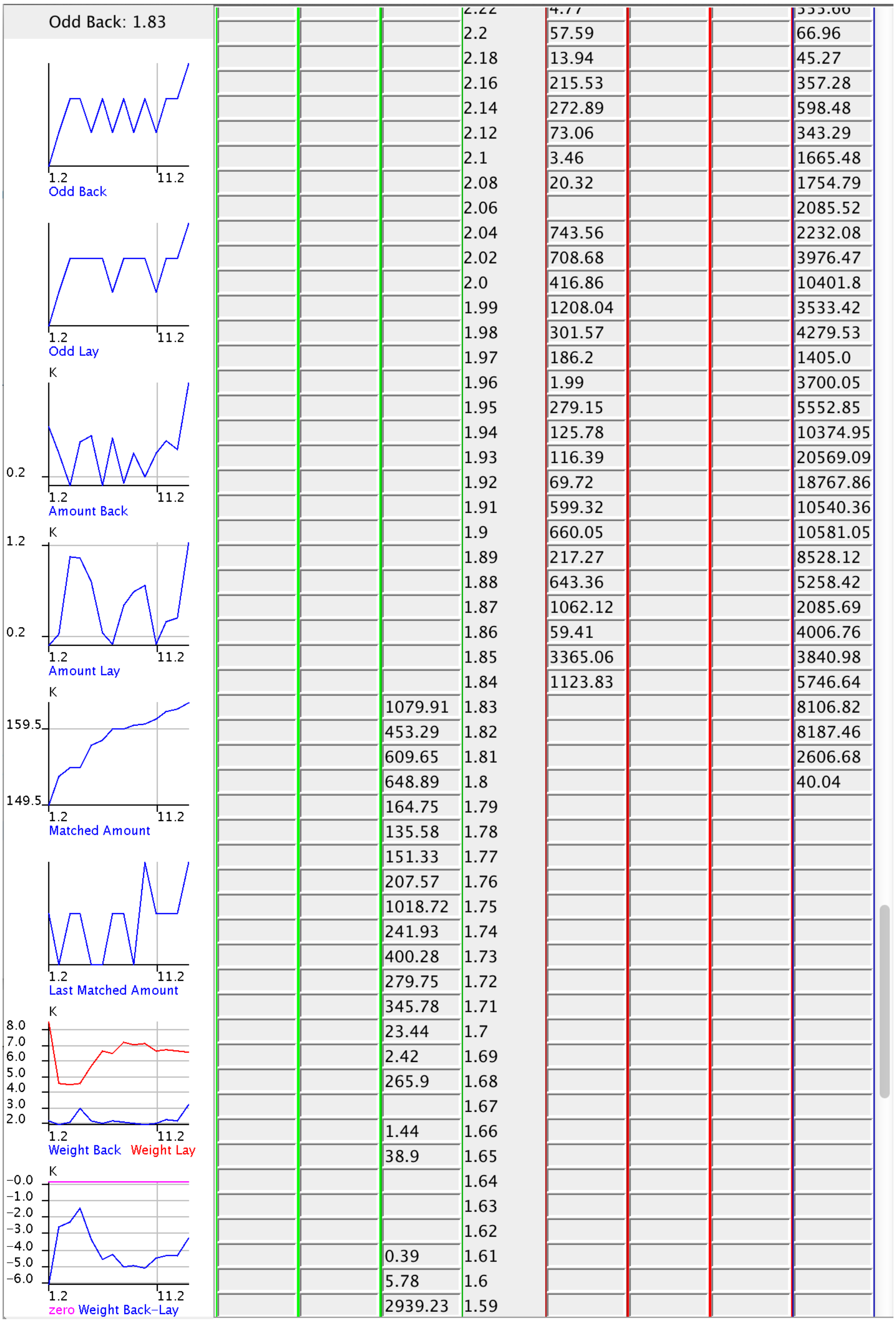
```
Profit/Loss:  0.9562229599724201
----- Dealer Statistics -----
[0] END_STATE : CLOSED
[1] EVENT : "DownP_13th_Jul"
[2] MARKET : "2m2f_INHF"
[3] RUNNER : "Keiths_Delight"
[4] RUNNER_MATCHED_AMOUNT : 1842.76
[5] NUMBER_OF_RUNNERS : 15
[6] STAKE : 10.0
[7] AXIS_ODD : 8.6
[8] TICKS_BACK_LOSS : 3
[9] TICKS_LAY_LOSS : 3
[10] REQUEST_ODD_BACK : 9.2
[11] REQUEST_ODD_LAY : 8.6
[12] STOPLOSS_ODD_BACK : 9.8
[13] STOPLOSS_ODD_LAY : 8.0
[14] TYPE : BACK
[15] MATCHED_ODD_BACK : 9.2
[16] MATCHED_ODD_LAY : 8.6
[17] MATCHED_AMOUNT_BACK : 10.0
[18] MATCHED_AMOUNT_LAY : 10.69767441860465
[19] PROFIT_LOSS : 0.6976744186046506
------------ || ------------
```

Figure A.1: Dealer GUI presenting statistics from execution of one trading mechanism dealer

Dealer GUI

Appendix B

Trading Tools



Odd Back: 1.83

			2.22	4.77			[illegible]
			2.2	57.59			66.96
			2.18	13.94			45.27
			2.16	215.53			357.28
			2.14	272.89			598.48
			2.12	73.06			343.29
			2.1	3.46			1665.48
			2.08	20.32			1754.79
			2.06				2085.52
			2.04	743.56			2232.08
			2.02	708.68			3976.47
			2.0	416.86			10401.8
			1.99	1208.04			3533.42
			1.98	301.57			4279.53
			1.97	186.2			1405.0
			1.96	1.99			3700.05
			1.95	279.15			5552.85
			1.94	125.78			10374.95
			1.93	116.39			20569.09
			1.92	69.72			18767.86
			1.91	599.32			10540.36
			1.9	660.05			10581.05
			1.89	217.27			8528.12
			1.88	643.36			5258.42
			1.87	1062.12			2085.69
			1.86	59.41			4006.76
			1.85	3365.06			3840.98
			1.84	1123.83			5746.64
		1079.91	1.83				8106.82
		453.29	1.82				8187.46
		609.65	1.81				2606.68
		648.89	1.8				40.04
		164.75	1.79				
		135.58	1.78				
		151.33	1.77				
		207.57	1.76				
		1018.72	1.75				
		241.93	1.74				
		400.28	1.73				
		279.75	1.72				
		345.78	1.71				
		23.44	1.7				
		2.42	1.69				
		265.9	1.68				
			1.67				
		1.44	1.66				
		38.9	1.65				
			1.64				
			1.63				
			1.62				
		0.39	1.61				
		5.78	1.6				
		2939.23	1.59				

Figure B.1: Ladder of JBet, with various analysis real-time graphics, to track trades made by agents