FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Building a Poker Playing Agent based on Game Logs using Supervised Learning

Luís Filipe Guimarães Teófilo

Mestrado Integrado em Engenharia Informática e Computação

Orientador: Professor Doutor Luís Paulo Reis

26 de Julho de 2010

Building a Poker Playing Agent based on Game Logs using Supervised Learning

Luís Filipe Guimarães Teófilo

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo Júri:

Presidente: Professor Doutor António Augusto de Sousa

Vogal Externo: Professor Doutor José Torres

Orientador: Professor Doutor Luís Paulo Reis

---

26 de Julho de 2010

Resumo

O desenvolvimento de agentes artificiais que jogam jogos de estratégia provou ser um domínio relevante de investigação, sendo que investigadores importantes na área das ciências de computadores dedicaram o seu tempo a estudar jogos como o Xadrez e as Damas, obtendo resultados notáveis onde o jogador artificial venceu os melhores jogadores humanos. No entanto, os jogos estocásticos com informação incompleta trazem novos desafios. Neste tipo de jogos, o agente tem de lidar com problemas como a gestão de risco ou o tratamento de informação não fiável, o que torna essencial modelar adversários, para conseguir obter bons resultados.

Nos últimos anos, o Poker tornou-se um fenómeno de massas, sendo que a sua popularidade continua a aumentar. Na Web, o número de jogadores aumentou bastante, assim como o número de casinos online, tornando o Poker numa indústria bastante rentável. Além disso, devido à sua natureza estocástica de informação imperfeita, o Poker provou ser um problema desafiante para a inteligência artificial. Várias abordagens foram seguidas para criar um jogador artificial perfeito, sendo que já foram feitos progressos nesse sentido, como o melhoramento das técnicas de modelação de oponentes. No entanto, até à data ainda não existe nenhum jogador artificial de Poker que consiga igualar os melhores jogadores humanos.

Este projecto de dissertação tem como objectivo a criação de um jogador de raiz através da observação de jogos, previamente realizados, entre jogadores humanos. Depois da obtenção do histórico dos jogadores, foram definidas e obtidas várias variáveis de jogo relevantes. Posteriormente foram usadas várias técnicas de aprendizagem para modelar o comportamento do jogador observado, nomeadamente classificadores baseados em aprendizagem supervisionada. Todos os classificadores foram testados e comparados utilizando métodos estatísticos que avaliam a capacidade de previsão do modelo gerado, nomeadamente a avaliação cruzada. Finalmente, foi produzido e testado um agente que usa a estratégia aprendida. Foi também criada uma Framework com o nome HoldemML que reproduz todos estes passos, para que qualquer pessoa consiga facilmente criar um agente através do histórico de jogos.

Assim, o resultado desta investigação é uma aplicação capaz de gerar um agente, recorrendo apenas ao histórico de jogos de um dado jogador. O agente gerado segue aproximadamente as tácticas presentes no histórico. Verificou-se que esta abordagem por si só é insuficiente para criar um agente competitivo, pois as estratégias geradas não conseguem vencer os agentes dotados da capacidade de modelar oponentes. No entanto verificou-se que uma estratégia que combine várias tácticas de diferentes jogadores, permite confundir mecanismos de modelação de oponente dos adversários, melhorando assim os resultados do agente. Por esta

razão, considera-se esta abordagem promissora, pois através da introdução de modelação de oponente nas heurísticas de mudança de táctica, deverá ser possível construir um agente competitivo.

Palavras-chave: Poker; Data mining; Modelação de oponentes; Aprendizagem Supervisionada; Inteligência Artificial

Abstract

The development of artificial agents that play strategic games has proven to be a worthwhile domain for research, being that many important computer science researchers have devoted their time to study games like chess or checkers, achieving notable results where the artificial agent surpassed the best human players. However, the research on stochastic games with imperfect information brings many different challenges. In this type of games, the agent must deal with problems like risk management, unreliable information and deception which make it essential to model the opponents to achieve good results.

In recent years, poker has become a mass phenomenon, and its popularity continues to increase. At the web, the number of players online has increased dramatically as well as the number of online casinos, making this a highly profitable industry. Also, due to its stochastic an incomplete information nature, Poker has proven to be a real challenging problem to the artificial intelligence, motivating its research. Many different approaches were followed to create a perfect Poker player and progress has been made towards this goal, like the improvement of opponent modelling techniques. However, to the date there is no artificial Poker player that matches the best human players.

This dissertation project aims to create an artificial poker player from scratch, watching past games between human players. After obtaining game history, relevant game variables were defined and extracted from data. Afterwards, to model the observed player behaviour, various machine learning techniques were used, more particularly, classifiers based on supervised learning techniques. All classifiers were tested and compared using statistical methods that evaluate the predictive power of the generated model, such as cross validation. Finally, an agent who uses the learned strategy was produced and tested. A framework named HoldemML that reproduces all this steps was created, so anyone can easily create a poker agent from game logs.

Therefore, this research work resulted is a complete application capable of generating a fully working agent, just by providing game logs. The generated agent approximately follows the tactics given by the provided data. It was found that this approach alone is insufficient to create a competitive agent, as generated strategies can't beat agents with the ability to model opponents. However it was found that strategy that combines several different tactics can confuse the adversary's opponent modelling mechanisms, thus improving the agent results. For this reason, this can be considered a promising approach, because it should be possible to build a competitive agent by introducing opponent modelling in tactic change heuristics.

Keywords: Poker; Data mining; Opponent Modelling; Supervised Learning, Artificial Intelligence

Acknowledgements

To make this work possible, all external collaboration was essential, which I would like to thank.

First, I would like to thank to Prof. Dr. Luís Paulo Reis, who proposed and supervised this work and also demonstrated great enthusiasm about it.

I would also like to thank to Faculdade de Engenharia da Universidade do Porto which was the institution where I started and completed my Master degree course which culminated in the realization of this work.

Finally I would like to thank to my family and friends. They always have supported me on the most difficult times.

Luís Filipe Guimarães Teófilo

Index

List of figures

List of tables

Abbreviations and Symbols

AI	Artificial Intelligence
ANN	Artificial Neural Network
API	Application programming interface
CPRG	Computer Poker Research Group
CPU	Central Processing Unit
DBMS	Database Management System
FEUP	Faculdade de Engenharia da Universidade do Porto
HTML	HyperText Markup Language
IRC	Internet Relay Chat
LIACC	Artificial Intelligence and Computer Science Laboratory
XML	eXtensible Markup Language
WWW	World Wide Web

1 Introduction

This chapter presents the context of this thesis work: theme of this thesis, motivation for this work, a summary, and goals to be accomplished as well as a brief description of what is explained in each chapter of the document.

1.1 Context

Artificial intelligence (AI) is the science and engineering of making intelligent machines [1]. Its applications are innumerable, from game playing or speech recognition to computer vision and expert systems. Its contribution to this technological society is very important.

At the beginning, the main objective of AI research was to develop a strong AI i.e. an intelligence that matches or exceeds human intelligence. As years went thought it was verified that a project like that was megalomaniac, so the AI research focused on solving particular problems, with that being called weak AI.

However, the processing/memory capacity of CPUs has doubled every 2 years, as specified in Moore's law [2]. If the hardware evolution continues at this rate, soon the computers capacity may exceed human brain capacity. This may lead the humanity to reach a technological singularity [3] i.e. the humans might be able to create machines that are more intelligent than man. Some projects like Blue Brain, which is an attempt to create a synthetic brain by reverse-engineering the mammalian brain down to the molecular level [4], clearly support the development of strong AI. These concepts are sensitive issues in the philosophy of human thought and religion. Even so, nowadays weak AI research has achieved far greater results than strong AI.

One of the fields with large focus in AI research is games. There are many games that were and continue to be a very interesting challenge for AI. Classic games like chess or checkers

proved to be a worthy challenge. Significant results with these games were achieved. One of the most notable/known works was Deep Blue, a computer that was able to defeat a chess world champion in a series of games.

Poker is game that is being a field of interest of AI research on the last decade. This game presents a radically different challenge compared to other games like chess. In chess, the two players are always aware of the full state of the game. This means that, although not computationally feasible, it is possible to represent a deep chess decision tree. Unlike that, Poker game state is hidden, each player can only see its cards or community cards, and therefore it's much more difficult to build and analyze a decision tree. Poker is also a stochastic game i.e. it admits the element of chance.

1.2 Motivation

There are various reasons that motivate Poker research. First of all, Poker is nowadays a growing industry, especially on the web [5]. Poker has also become a very popular game, especially Texas Hold'em variant, even receiving media coverage, like popular sports. One good example of that is the Poker Channel [6] which is a TV channel dedicated exclusively to the transmission of poker news, important poker events, etc.

Besides being a recent popular game and having a growing market, Poker also represents a very different challenge in AI that therefore motivates the research in this area. The main aspect that distinguishes Poker from most other games is its imperfect information nature where certain relevant details are withheld from the players, or where knowledge is not reliable [7]. This aspect cannot be ignored when defining a strategy, since handling the very little available information is fundamental to competent play. Combining this with Poker rules, which are simple and have well defined parameters as well as factors like risk management, necessity of bluffing, implications of multiple opponents, discerning deception, and deducing the styles of other players in order to exploit their weaknesses [7], make Poker a rich a challenging game to computer science.

Besides the scientific interest to computer science, Poker may also have importance to other knowledge areas. For instance, Poker, like other gambling games, can have a great economic impact [8]. Poker has also been abroad by psychologists who have studied the relation between psychology and opponent modelling [9,10] having reached conclusion that the player behavior changes after big wins and losses, most properly, after a big pot loss most players tend to play less cautious [10].

1.3 Goals

In this thesis, the game of Poker will be analyzed from the standpoint of AI. There is a lot of research about Computer Poker, so this thesis will focus on a particular set of research. The research will focus on how strategies used in past human games can be used to support the creation of an artificial poker player agent.

The main goals of this thesis work are:

- Extract a significant amount of poker game history data from game logs between human players.
- Define game variables that can characterize a tactic from the extracted game data i.e. by each game state, in game history, define which variables can influence the human player decision.
- Build a strategy from extracted game variables from the players' game history, using machine learning namely supervised learning techniques.
- Create a poker playing agent that uses previously learned strategies.
- Discover if the learned strategies behave as expected and if they can be considered competitive, using simulation software.
- Discover to what extent human strategies can be used in order to assist artificial poker playing agent decisions.
- Build a framework to generate and test strategies built from game logs between human players.

1.4 Summary of contents

This document is divided into eight chapters.

In the first chapter it is presented the thesis theme as well as the context of its realization, motivation and definition of generic goals to be achieved.

The second chapter contextualizes the domain of the problem and its importance for the artificial intelligence researcher’s community. It is presented a quick overview about No Limit Texas Hold'em Poker, by explaining its rules.

The third chapter presents the state of art about tools and research that were useful for this thesis work. In this chapter, each sub chapter is related to one of the phases of implementation of this work.

The fourth chapter presents the whole process that was followed to accomplish this thesis goals, by explaining briefly each development phase of this thesis, the global architecture of the

developed agent and the types of testing that were applied to the agents as well as the global architecture of the HoldemML Framework.

The fifth chapter presents the data extraction and data normalization phase of this thesis work i.e. the steps that were needed to extract history data from games between human players. It is presented each application developed for this outcome (HoldemML Converter, HoldemML Validator, HoldemML Player list Extractor and HoldemML Stats Extractor) as well as the specification of HoldemML Schema format.

The sixth chapter presents the strategy building phase of this thesis work. It will be presented all the classifiers used to learn human tactics and how are they combined to form a strategy. Each application developed for this outcome (HoldemML Strategy Generator, HoldemML Bot and HoldemML Simulator) are also presented.

The seventh chapter presents all the tests on the generated agents, comparison between tactics and strategies, agent behavior analysis and tests against other artificial poker agents developed by other researcher's.

The eighth and final chapter presents the conclusions on research done on the subject and future work perspectives.

2 Poker

Poker is a common denomination for of card betting games with similar rules. In this chapter it will be presented a quick approach to poker, including the variant that is object of study: No Limit Texas Hold'em.

2.1 Incomplete information stochastic games

Strategic games can be classified by two parameters: deterministic and information completeness.

A game by being deterministic means that the next state of the game will be purely defined by the current player, without external factors. If a game is not deterministic, it means that there external factors influence player decision, such as random events. A game where random events influence the game flow can be also called stochastic.

As regards to information completeness, a game with complete information is a game that anyone, at any stage of the game, can identify the whole state of the game. A game with incomplete information, partial information of the game state is not known by every player, which means that it's not always possible to know the consequences of an action.

Figure 2.1 shows examples of various types of games.

Poker

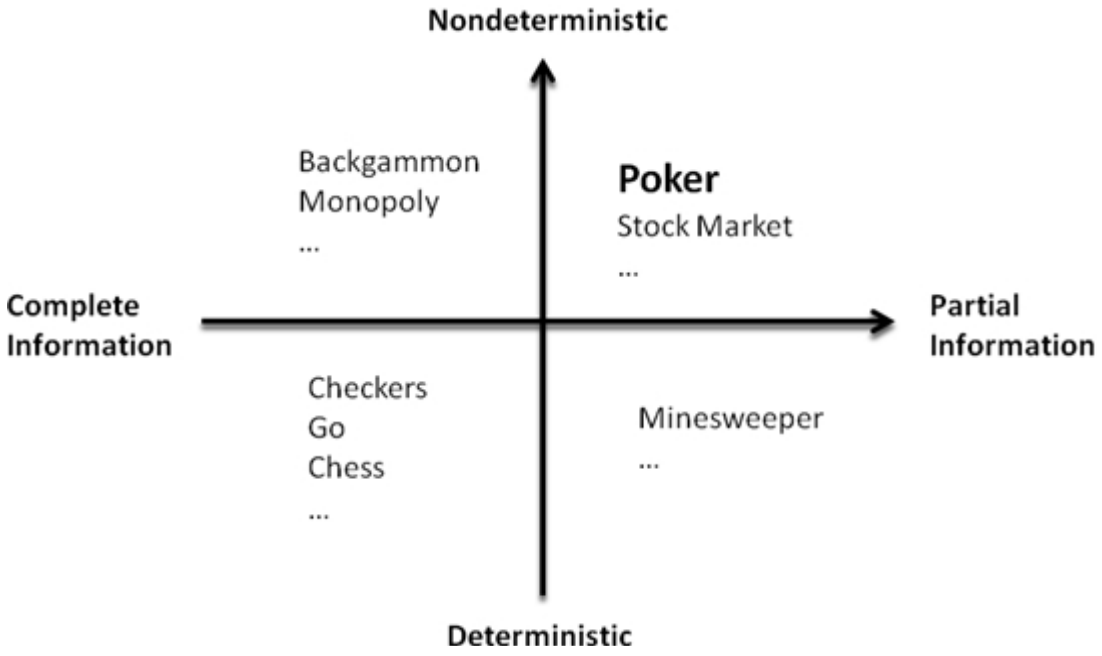

Fig 2.1: Game classification

As it can be seen, Poker is a non deterministic (stochastic) game with partial information. Poker is a game with partial information because the players don't know which cards their opponent's have, and it is stochastic because the pocket and community cards are randomly distributed.

2.1.1 Importance to artificial intelligence

The area of computer strategic game playing has a history almost as long as computer science itself. Strategic games have proven to be a worthwhile domain for study, and many prominent figures in computer science have devoted time to the study of computer chess and other skill-testing games. Some of these luminaries include John Von Neumann, Alan Turing, Claude Shannon, John McCarthy, Donald Knuth, Alan Newell, Herbert Simon, Arthur Samuel, Donald Michie, and Ken Thompson [7].

Poker is important for artificial intelligence as it presents a completely new challenge compared to other games like chess. To play Poker a player must make predictions of other players, with incomplete information about them. These mechanisms to deal with misinformation are a complex challenge. With proper adjustment they can be used in other branches of science, to process fuzzy information, and make predictions with little knowledge of the state of the domain.

2.2 Poker Overview

Poker is a generic name for literally hundreds of games, but they all fall within a few interrelated types [11].

It is a card game in which players bet that their hand is stronger than the hands of their opponents. All bets go into the pot and at the end of the game, the player with the best hand wins. There is another way of winning the pot that is making other players forfeit and therefore being the last standing player.

2.2.1 Hand Ranking

A poker hand is a set of five cards that identifies the strength of a player in a game of poker. It is not necessary to have all the cards belonging to one player, because in poker there is the definition of community cards – cards that belongs to all players. In poker variations that use community cards, the player hand is the best possible hand combining his cards with community cards.

The following card hands and descriptions represent the poker hand ranking, in descending order of strength.

Royal Flush: this is the best possible hand in standard five-card poker. Ace, King, Queen, Jack and 10, all of the same suit.

Fig 2.2: Royal Flush example (Hearts)

Straight Flush: Any five-card sequence in the same suit.

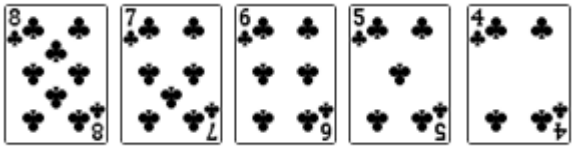

Fig 2.3: Straight Flush example (Clubs)

Four of a kind: All four cards of the same value (or rank).

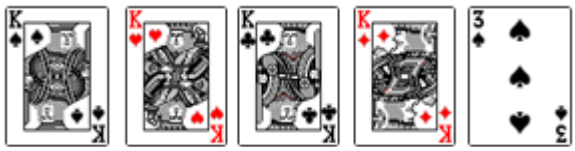

Fig 2.4: Four of a kind example (King)

Full House: Three of a kind combined with a pair. It is ranked by the trips (three of a kind).

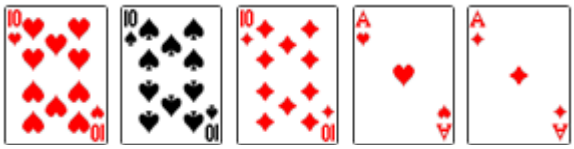

Fig 2.5: Full House example (10-Ace)

Flush: Any five cards of the same suit, but not in sequence. It is ranked by the top card.

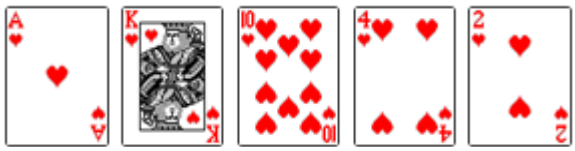

Fig 2.6: Flush example (Hearts)

Straight: Five cards in sequence, but not in same suit. The Ace plays high or low in straight. It is ranked by the top card.

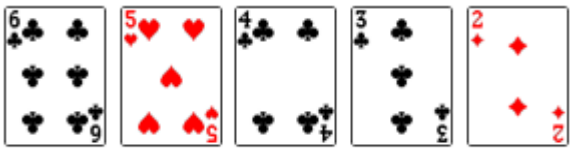

Fig 2.7: Straight example (Six High)

Three of a kind: Three cards of the same value.

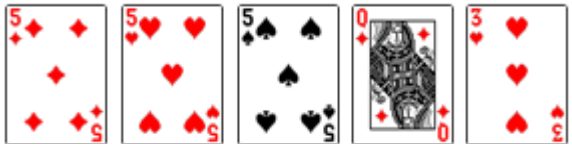

Fig 2.8: Three of a kind example (Five)

Two pair: Two separate pairs, and one kicker of different value. The kicker is used to decide upon a tie of the same two pairs.

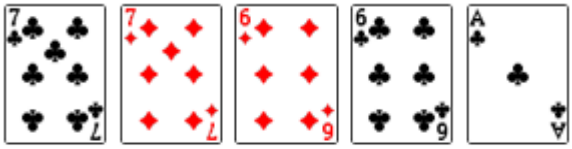

Fig 2.9: Two pair example (Seven/Six)

One pair: Two cards of the same value. Three kickers.

Fig 2.10: Pair example (Ace)

High card: Any hand that does not qualify as one of the better hands above. Ranked by top card, then the second card and so on.

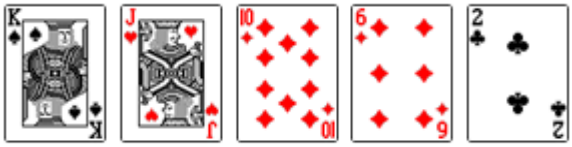

Fig 2.11: High card example (King)

2.2.2 No Limit Texas Hold'em

No Limit Texas Hold’em is a Poker variation that uses community cards.

At the beginning of every game, two cards are dealt for each player. A dealer player is assigned and marked with a dealer button. The dealer position rotates clockwise from game to game. After that, the two players to the left of dealer post the blind bets. The first player is called small blind, and the next one is called big blind. They respectively post half of minimum bet and a minimum bet. The layout of the poker table can be observed in figure 2.12. The dealer is the player in seat F and seats A and B are respectively small blind and big blind players.

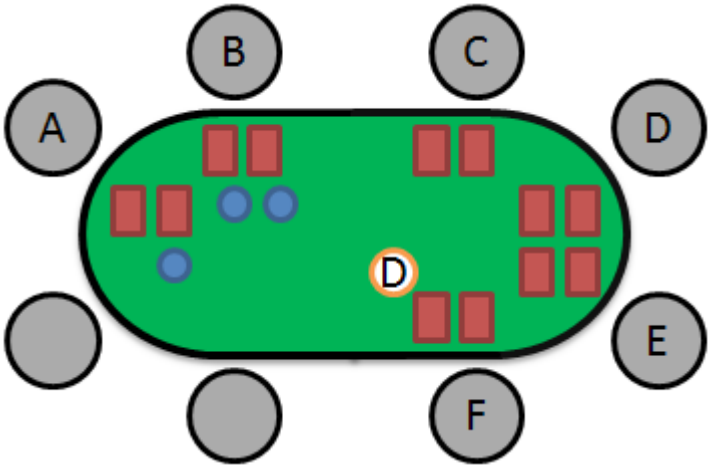

Fig 2.12: Poker Table Layout

The first player to act is the player on the left of the big blind. The next player is always the one to the left of the current player. Each player can execute one of the following actions:

- **Bet**: put money in the pot;
- **Call**: match the current highest bet;
- **Raise**: put a bet higher than the current highest bet;
- **Check**: if no bets were made, it's possible to pass to the next player without putting any money on the table;
- **Fold**: forfeit the cards and thus giving up the pot.
- **All-In**: a special type of raise. In this type of raise the player puts his entire bankroll on the table.

A player to continue to discuss the pot is obliged to call or raise the maximum current bet. In no limit Texas Hold'em there is no bet limit, therefore the value of the bet can go from the minimum bet (blind) up to the full bankroll of the player.

After all players go all-in, call or check, a round is finished. There are four betting rounds at Texas Hold'em. In each round, new community cards are revealed. The four rounds are:

- **Pre-Flop**: no community cards;
- **Flop**: three community cards are dealt;
- **Turn**: the forth community card is dealt;
- **River**: the fifth and final community card is dealt.

After the river, if all players agree to call/check the pot, it's time for the **showdown**. In the showdown every player shows its cards, and the one with the best hand wins the pot. The hand rank of any player is the best possible 5 card hand rank of the combination of his pocket cards and community cards. If two players have similar ranked hand, there is a tie and the pot is divided.

In each round if all players fold except one, the remaining player it the winner and can collect the pot.

There are also other specific rules of this variation that are not going to be detailed here, such as the existence of side pots when someone goes all-ins.

2.3 Summary

In this chapter it was introduced the problem domain through a fast approach to the importance of poker in artificial intelligence and the variant of poker that will be the subject of this dissertation study.

3 State of the art

In this chapter it will be presented the state of art about each step of development of this thesis work. Most of the information present on this state of art is based on research done by University of Alberta Computer Poker Research Group (CPRG) [13] which has published the most renowned works about AI applied to Poker.

3.1 Overview

Most noticeable achievements in this area are from the Poker Research Group [13] in the University of Alberta, having won several Computer Poker Tournaments [7, 14, 15, 16, 17, 18]. Most of the group's published work was on Fixed-Limit Hold'em Poker, however, many ideas might be used in variant no limit with some adjustments. Most renowned group's publication is Darse Billings PhD thesis [7] which provides a complete analysis of the evolution of artificial poker agent architectures, demonstrating strengths and weaknesses of each architecture both in theory and in practice. Other important publications were: Morgan Kan's Master thesis [14], which presents DIVAT, a tool for analysing the quality of agent decisions in Poker; "The Challenge of Poker" [19], an article which presents the architecture of Poki agent, one of the best to date; "Opponent modelling in poker" [15], a master thesis which introduced a new method called miximax to search game trees and provide a reliable estimate of opponent actions.

Other significant publications at AAMAS [20, 21] and AAAI [22] international conferences in this domain are those by Tuomas Sandholm from Carnegie Mellon University. His research aims to create a perfect strategy, which was achieved in a very simple variant of Poker (Rhode Island). Another near-perfect strategy was achieved by Miltersen and Sorensen [23] in Heads-Up No Limit variant, using the Nash Equilibrium Theory.

More recently, it should be emphasized the articles: “Data Biased Robust Counter Strategies” by Johanson and Bowling [24], which describes a new and more effective technique for building counter strategies based on Restricted Nash Response; “Strategy Grafting in Extensive Games” by Waugh and Bowling [25], which explains how to distribute the process of solving equilibria in extensive games like Poker.

Some achievements were also made using pattern classifiers, in articles like “Pattern Classification in No-Limit Poker: A Head-Start Evolutionary Approach” [26], which used evolutionary methods to classify in game actions; or “Using Artificial Neural Networks to Model Opponents in Texas Hold´em” [18], where is demonstrated that Artificial Neural Networks can be used to model opponents.

Great deals of opponent modelling techniques are based on real professional poker players' strategies, such as David Slansky, who published one of the most renowned books on poker strategy [11].

Despite all the breakthroughs achieved by known research groups and individuals, no artificial poker playing agent is presently known capable of beating the best human players. In the following pages it will be reviewed research on the most relevant aspects for this thesis including previous work and supporting tools that might be useful for studies in this matter.

3.2 Collecting game data

To model a strategy the first thing needed is to obtain data from games between human players. There is a large amount of game data available on web however most data is owned by online casinos therefore it’s not free to use. Furthermore, there are some desirable requisites that must be accomplished in order to ensure data quality. For instance, quality data sources should have:

- Large amount of games of each individual. To learn the strategy of play from a certain player it’s mandatory to have lots of info of that player. The more info about the player, a more accurate strategy representation is reachable.
- Real money games. In virtual money poker there are lots of players that easily risk their chip stacks, because they have nothing to lose. In real money poker, players besides risking their money, which they are not willing to lose, can actually win a prize-money. This means that the money factor is usually a strong motivation that causes players to be more careful which results in more complex strategies. These facts are suggested in [9, 10].

After doing some research, various data sources were found that may be useful to use in this thesis work.

3.2.1 BWIN Database

BWIN Interactive Entertainment AG is an Austria-based provider of online gaming entertainment. It offers sports betting, poker, casino games and soft and skill games [27].

Major online Poker sites apparently don't allow obtaining information of the hands as spectators [28]. However though BWIN it's possible to retrieve some poker game data in HTML. BWIN does not provide any tool to save the information directly in other formats; even so it's possible to transform the web page that contains hand information [28]. The following table shows all necessary steps to extract poker game data and store in a database.

Table 3.1: BWIN to database download and conversion process (adapted from [28])

Step 1: Log on to bet and win
Step 2: Select the table
Step 3: Download the hands
Step 4: Join the hands
Step 5: Convert from html to txt
Step 6: Convert to database

The problem of this process is that it's quite time consuming. To extract about 40.000 hands about 13 hours are needed. Besides that, only last week hands are available which means that this process would take a few weeks to get those 40.000 hands.

3.2.2 Michael Maurer's IRC Poker Database

Michael Maurer's IRC Poker Database is a big database of poker games played in IRC channels before the advent of real-money online poker servers [29], hosted by University of Alberta Computer Poker Research Group [13].

This database is a collection of more than 10 million hands and it was recorded from the player perspective. In other words the cards of opponent folded hands are hidden. Despite the larger size of the database, it only includes virtual money games which reduce the database value. However due to lack of real money online competition during 1995-2001, some players

are good, especially the ones from higher tiered IRC games that required larger bankrolls to qualify. Some of these players were regular 20-40 casino players [29].

3.2.3 PokerTracker

PokerTracker is a suite of software products that tracks the user during online play [30]. This software consists in two applications:

- PokerTracker: software capable of downloading hand history from compatible online casinos.
- TableTracker: software that uses an internal database of online players and indicates which tables are best to win money i.e. the ones where the players are weaker.

After testing this application in a couple of online casinos, it was verified that only user hand history could be obtained. The data obtained was based on logs stored by the casino client. This data could be useful to track the user hand history, so the user can know his limitations and improve his playing. For the TableTracker since it's not possible to access the internal database, the application is useless for this thesis work.

3.2.4 FreePokerDB

Freepokerdb is an application made in python, similar to PokerTracker, that is able to track user online poker games, the behaviour of the other players and user winnings/losses [31]. The main advantage over PokerTracker is that Freepokerdb is a free to use and open source application. So far it supports well known online casinos such as PokerStars [32] or Full Tilt Poker [33]. The application is still under development so new features may arise in future.

3.2.5 FlopTurnRiver.com Hand Database

The website flopturnriver.com provides a database of over 800.000 hands of poker. These hands are categorized by a range of parameters like stakes, pot value, game structure (no limit/limit), poker game variation (Omaha and Hold'em) or player position. These hands were extracted from popular online casinos. The fact that most hands represent money games makes this valuable moreover they are free to use. The only problem about this database is extraction. The provided data is available in HTML pages which mean that to extract data an HTML parser is needed.

3.2.6 PokerFTP Database

PokerFtp website [34] has a database of nearly one billion real money poker hand histories, played on some major online casinos between 2006 and 2008. PokerFtp has also available software to access this data.

To get access to the full database, it is necessary to develop a small application that uses a smaller version of the database, download-able in Poker Ftp website.

3.2.7 Online Casino Logs

There are individual poker players that provide their casino client logs. However the format usually lacks structure which means that is time consuming to build a parser that interprets these files.

3.3 Representing game data

In a research work like this thesis, it's very common that data may come from different sources with different representations of poker hand history. Due to this fact, it is necessary to adopt a common data format to simplify the data processing stage.

A proper format for representing poker game data is essential. One of the goals of this work is to use format standards. Using standard formats is a good practice because there are more pre developed tools to support file parsing and data portability increases. Besides that it improves data reusability which means that the data used in this thesis could be reused in other research works.

After doing some research, no standard file format for representing hand history poker data was found. There are many formats for representing poker data. In fact most online casinos have their own representation. Unfortunately, most representations are undisclosed, and some are probably just object serializations or text based files without any structure.

Due to these facts, the better solution might be the development of a new representation. A good choice for the definition of the new poker hand history representation is to use XML [35]. XML, which stands for Extensible Markup Language, is a very simple and widely used document annotation and description language. It has many advantages over the use of another type of language like the considerable quantity of processing tools, interoperability, portability, simplicity and even more advantages not mentioned [36].

Other criteria that favours choosing XML is its Database Management System (DBMS) support. In research works like this one, typically lots of data are used. To store and quickly access data, the use of a relational databases is essential. Nowadays most DBMS supports XML processing and table to element mapping.

3.3.1 Online Casino Logs

As explained earlier, the casino logs typically contain no structure, making it difficult to interpret the information contained. Moreover, each casino client has its own representation of game data and that is why it's necessary to create a parser for each format to combine data from different sources.

For instance, a game example from Fult Tilt Poker [33] is shown bellow.

```
Full Tilt Poker Game #361920714653929: Table stars - $0.05/$0.10 -
No Limit Hold'em - 07:04:49 BST - 2010/06/20
Seat 1: Joe ($10.00)
Seat 2: Mathew ($10.00)
Mathew posts the small blind of $0.05
Joe posts the big blind of $0.10
The button is in seat #1
*** HOLE CARDS ***
Mathew raises to $9.00
Joe raises to $9.01
Mathew raises to $9.90
Joe raises to $9.90
Mathew raises to $10.00
Joe calls $0.10
Mathew shows [8s 7d]
Joe shows [7c 9h]
*** FLOP *** [9d 2h 6d]
*** TURN *** [9d 2h 6d] [2s]
*** RIVER *** [9d 2h 6d 2s] [Qh]
Joe wins the pot ($20.00)
*** SUMMARY ***
Joe showed [7c 9h] and won ($20.00)
```

Fig 3.1: Full Tilt Poker Log example

As it can be seen, the format of Full Tilt Poker logs is text based therefore more appropriate for human reading.

3.4 Strategy representation

After learning some strategies from poker data, the next step is representing them in a file format or database. Some languages that represent a poker strategy were already defined and can approximately represent a static poker strategy with some opponent model features based on opponent pre-defined classifications.

3.4.1 Poker Programming Language

Poker programming language (PPL) is the default language to customize Shanky Technologies Poker Bots. With it a user can fully customize the play of Shanky bots, whether it is making a few adjustments or creating an entire set of playing instructions from scratch [37].

The structure of PPL documents is the following:

- Starting line with the string “custom” which means that the document represents a custom strategy;
- Line with string “PreFlop”, which indicates the beginning of pre flop strategies specification;
- Strategy statements (one per line), which structure will be explained later;
- Repetition of the two last steps for “Flop”, “Turn” and “River” rounds.

An example of a PPL document may be found in figure 3.2.

```
custom

Preflop
When hand = AA RaiseMax force
When hand = KK RaiseMax force
When hand = QQ RaiseMax force
When hand = JJ RaiseMax force
When hand = AK RaiseMax force
When hand = AQ RaiseMax force
When others fold force

Flop
When (BotsLastAction = beep or BotsLastAction = check) beep force

Turn
When BotsLastAction = beep beep force

River
When BotsLastAction = beep beep force
```

Fig 3.2: Poker Programming Language example

In this example strategy, the bot when getting premium hands like top pocket pairs or Ace King or Ace Queen, it goes all-in.

The strategy statements are structured as follows.

```
Round {Preflop, Flop, Turn, River }

When <Condition> <Action> or When <Condition>

When <Condition> <Action> or When <Condition>

When <Condition> <Action> or When <Condition>
…
```

Fig 3.3: Poker Programming Language structure

3.4.1.1 Conditions

There are four types of conditions:

- Hand specification condition: This condition specifies the cards that the player has. Examples: Hand = A J, Hand = A K suited;
- Board specification condition: This condition specifies the table cards. It only makes sense in post-flop actions. Examples: Board = 3 4 A;
- Variable Comparison Condition: This condition is used to compare a predefined variable to a numeric value. Examples: Raises < 2, Folds > 2;
- Relative pot size / stack size comparison condition: This is a special condition designed to allow you to compare the percentage of a certain numeric valued variables to either pot size or stack size. Example: BetSize < 30% PotSize.
- Position conditions: Condition that checks for bot table position. Examples: In BigBling, StillToAct = 2, Position = Last.

Conditions can also be combined with OR/AND logic operators. Conditions can also be expressed as random such as “WHEN RANDOM > 75” specifies that the condition is true 75% of times.

3.4.1.2 Actions

PPL supports the following actions:

- Beep: action is done by player;
- Call: the agents calls the highest bet or goes all-in if it does not have enough money;
- Play: the agent check of fold;
- Raise: the agent raises a given value;
- RaiseMin: the agent raises the minimum possible value (big blind value);
- RaiseHalfPot: the agent raises half of its money;
- RaisePot: the agent raises a given percentage of its pot;
- RaiseMax: the agent raises its complete pot;
- Fold: the agent folds the hand;
- Bet: the agent bets a given value;
- BetMin: the agent bets the minimum possible value (big blind value);
- BetHalfPot: the agents bets half of its pot;
- BetPot: the agent bets a given percentage of its pot;
- BetMax: the agent bets its complete pot;
- SitOut: tournament or poker room exit.

3.4.2 PokerLang

PokerLang is a high level language of poker concepts what permit to create simple rules to define a poker agent [38].

This language is rather simple as it is indented to be used by common users, so anyone can specify a poker playing strategy. This fact may pose serious problems in strategy definition flexibility. This can be solved by expanding or adapting the language definition.

The following figure shows the PokerLang main concepts.

```
<STRATEGY> ::= { <ACTIVATION_CONDITION> <TACTIC> }
<ACTIVATION_CONDITION> ::= { <EVALUATOR> }
<TACTIC> ::= {PREDEFINED_TACTIC | <TACTIC_NAME> <TACTIC_DEFINITION>
<PREDEFINED_TACTIC> ::= loose_agressive | loose_passive | tight_agressive | tight_passive
<TACTIC_NAME> ::= [string]
<TACTIC_DEFINITION> ::= { <BEHAVIOUR> <VALUE> }
<BEHAVIOUR> ::= { <RULE> }
<RULE> ::= { <EVALUATOR> | <PREDICTOR> } <ACTION>
<EVALUATOR> ::= <NUMBER_OF_PLAYERS> | <STACK> | <POT_ODDS> |
<HAND_REGION> | <POSITION_AT_TABLE>
<PREDICTOR> ::= <IMPLIED_ODDS> | <OPPONENT_HAND> | <OPPONENT_IN_GAME> |
<STEAL_BET> | <IMAGE_AT_TABLE>
<ACTION> ::= { <PREDEFINED_ACTION> <PERC> | <DEFINED_ACTION> <PERC> }
<PREDEFINED_ACTION> ::= <STEAL_THE_POT> | <SEMI_BLUFF> |
<CHECK_RAISE_BLUFF> | <SQUEEZE_PLAY> | <CHECK_CALL_TRAP> |
<CHECK_RAISE_TRAP> | <POST_OAK_BLUFF>
```

Fig 3.4: PokerLang main definition

A PokerLang strategy is composed by tactics, each one having an activation condition. The activation condition is based on a table evaluator, which uses the table information to determine if the tactic shall be used. The chosen tactic could be a predefined tactic or a tactic defined in the PokerLang document, which is composed a set of rules. Each rule uses an evaluator and a predictor to determine if a given action should be followed.

3.5 Strategy learning

After having ready all poker data, the next step is to process it. Strategy learning means to search out player strategy from hands of data player.

There are several approaches to learn strategies, mainly based on player or opponent modelling techniques. In the following lines, some of these approaches will be demonstrated.

3.5.1 Strategy modelling approaches

To create a good poker strategy three main approaches can be followed:

- Nash Equilibrium strategies;
- Best response strategies;
- Restricted Nash Response – combination of the last ones.

3.5.1.1 Nash Equilibrium strategies

Nash equilibrium is a concept of game theory that proves that for every multiplayer game there is a set of strategies that, even remaining static, always preserve the same advantage over the opponents even if the opponents change strategy. Due to the high number of variables present in Texas Hold'em, these types of strategies are very difficult to model so they only apply in certain game situations [39]. For instance, SAGE System, which was developed based on game theory, is an approximation of Nash equilibrium to play heads-up in Texas Hold'em. The system is not flawless, however against most opponents, who don't correctly adjust for the large blinds and weaker hand values of heads-up play; the user is given an advantage of about 5% to 40% [40].

3.5.1.2 Best response

Best response theory consists in the paradigm that for every strategy, there is the best possible counterstrategy [28]. Since the best players change strategy during the game, an agent must recognize these changes during play in order to be competitive. The problem of this type of strategies is that at the start of the game there is possibly no knowledge about the opponent, besides that, learning about an opponent need a considerable amount of information. For instance in experiments with BRPlayer against PsOpti4, BRPlayer took between 20,000 to 175,000 hands before it was able to break even [7].

3.5.2 Player modelling

Player modelling consists in characterize the game variables and constraints in order to represent a strategy. According to João Ferreira [28] the following variables can describe the strategy on a particular game:

- <Game> ::= <Table> <Players> <History>
- <Table> ::= <Number Players> <Blind> <Fold Ratio> <Average Pot> <Time>
- <Players> ::= {<Player>}
- <Player> ::= <Name> <Position> <Money Flotation>
- <Position> ::= Green Zone | Yellow Zone | Red Zone

- <History> ::= <Game> <Actions>
- <Game> ::= <Starting Player> <Player Card Strengths> <Board Card Strength>
- <Player Card Strengths> ::= <Player Card Strength>
- <Player Card Strength> ::= 1..9
- <Actions> ::= <Pre Flop> <Flop> <Turn> <River>

It's possible to simplify this model to analyze player behaviour on a certain round of the game. João Ferreira [28] defined a simplified model to analyze player behaviour on pre-flop round. His conclusions were that aggressive actions in pre-flop work better than passive actions and players change their strategy based on number of table players, player actions and his position.

Another possible approach for player modelling is the combination of neural networks with evolutionary methods such as genetic algorithms, know as evolutionary neural networks. The idea behind this is raising a population of neural networks, each representing an agent. Agents participate in tournaments consisting up to 2000 agents for 500 generations. After each generation, the agents of performed best in the tournaments are selected and retained [42]. After that, the best agents are combined to create a child agent. After giving a normally distributed random bias to each parent, the weights of child agent are calculated based on the following equations 3.1 and 3.2.

$$C_x = B_0 * (P_0)_x + B_1 * (P_1)_x + \ldots + B_n * (P_n)_x \tag{3.1}$$

$$C_x = \sum_{(i=0)}^{n} (B_i * Pi_x) \tag{3.2}$$

In this equation B's are each parent random bias while P's are parent weights. A mutation operator is also used. This mutation operator introduces some normally distributed random noise in a defined percentage of child agents, while the best parents remain unaltered. The input features of the neural network are described in the following table.

Table 3.2: Input features of the evolutionary Neural Network [42]

Feature	*Description*
Pot	Chips available to win
Call	Amount needed to call the bet
Opponents	Number of opponents
Win Percent	The percentage of hands that will win
Chips	The chip counts for each player
Totat	The overall aggressiveness of each player
Current	The recent aggressiveness of each player

3.5.3 Opponent modelling

Opponent modelling is an important step for creating a good poker agent. Opponent modelling means learning other opponent's strategy while playing against them. This is rather important because if a player uses a static strategy, it will be easier for his opponents find his play style making the player easy to defeat him. For this reason a good player use complex strategies that are difficult to learn and, from time to time, he changes his strategy to make even harder for his opponents to estimate his actions. The main goal of opponent modelling is to quickly find out which strategies the opponents are following and to detect strategy exchange.

A great deal of research has been done in opponent modelling area as it seems to be a key feature in poker agents.

João Ferreira has done statistical analysis of player models from multiplayer and concluded that learning opponent's behaviour from their past games can be a significant advantage for future games [28].

Dinix Félix had done some interesting experiments with opponent modelling. He created 8 agents, each with a different static strategy. Then he created an observer agent. After simulating about 10.000 games against each one, he concluded that an observer agent has better results than a non observer agent [43, 82, 83].

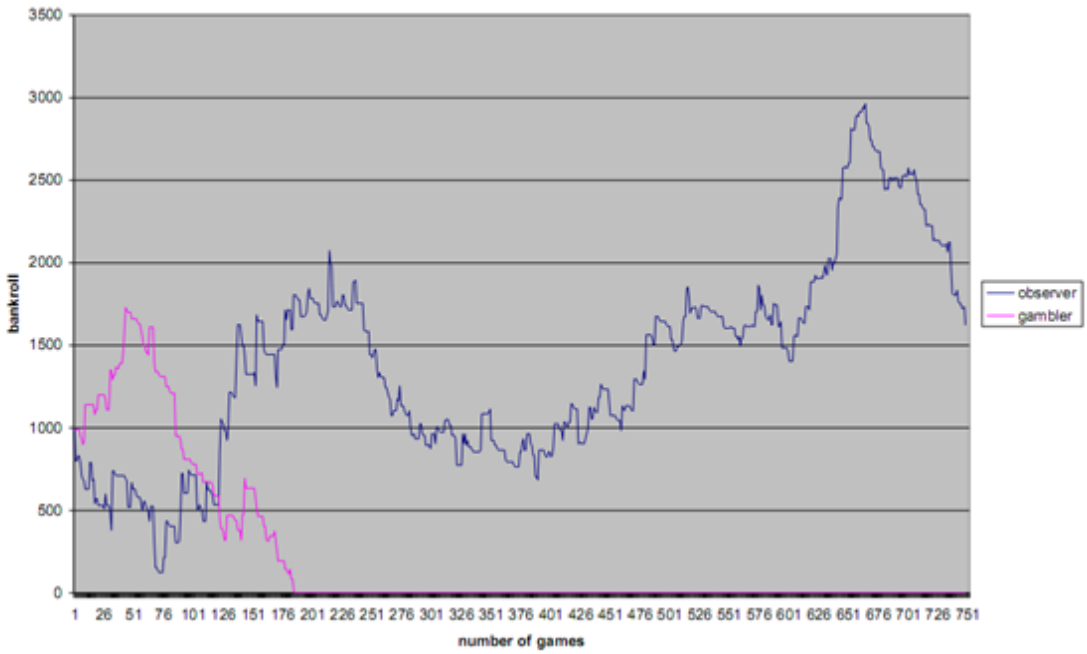

Fig 3.5: Example of simulation results in a match between an observer agent and gambler agent [43]

Neural Networks can also be used for opponent modelling. After some experiences done by Aaron Davidson, it was verified that neural networks can be pretty accurate to predict

opponent post flop actions, especially for static and generalized opponent modelling. Good results in dynamic opponent modelling can also be achieved by training not one network per player, but several [18]. The following table and figure shows examples of game variables to use in neural networks for opponent modelling.

Table 3.3: Context information used to train opponent modelling neural networks [18]

#	*Type*	*Description*
1	Real	Immediate Pot Odds
2	Real	Bet Ratio: bets/(bets+calls)
3	Real	Pot Ratio: amount_in / pot_size
4	Boolean	Committed in this round
5	Boolean	Bets-to-call == 0
6	Boolean	Bets-to-call == 1
7	Boolean	Bets-to-call >= 2
8	Boolean	Stage == FLOP
9	Boolean	Stage == TURN
10	Boolean	Stage == RIVER
11	Boolean	Last-Bets-To-Call > 0
12	Boolean	Last-Action == BET/RAISE
13	Real	(#players Dealt-In / 10)
14	Real	(# Active Players / 10)
15	Real	(#Unacted Players / 10)
16	Boolean	Flush Possible
17	Boolean	Ace on Board

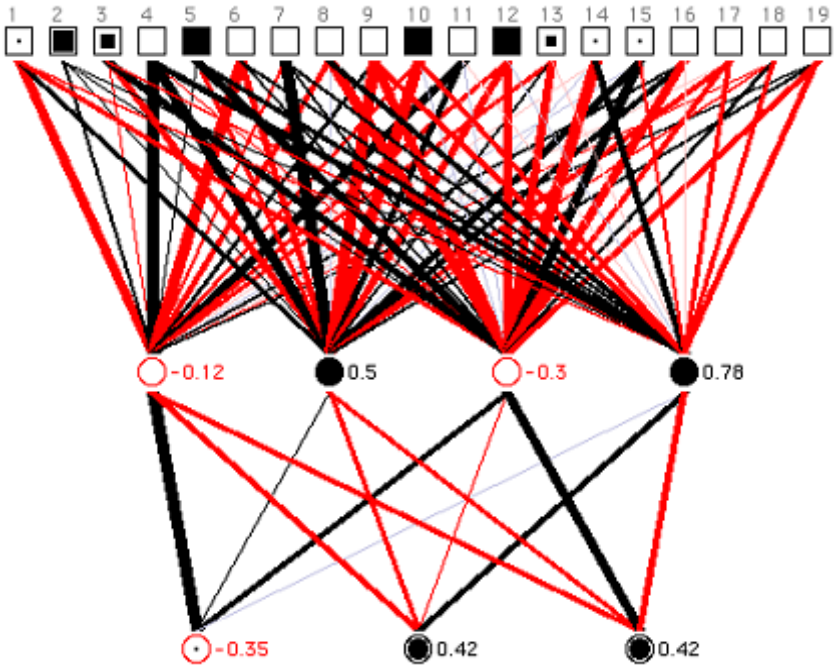

Fig 3.6: Opponent modelling neural network setup[18]

Decision trees can also be used for opponent modelling. Patrick McNally and Zafar Rafii analyzed different action rates (check rate, fold rate …) and the proportion between aggressive actions and passive actions thought the use of decision trees. The results were compared with neural networks and it was concluded that decision trees were much more accurate in pre-flop round while neural networks were slightly more accurate in the other game rounds. However the networks were trained for a very brief number of epochs and better performance is likely possible [44].

3.5.3.1 Slansky classification

Player classification is an important step for opponent modelling. It consists in grouping players with similar play styles. The criteria used for player classification is based on how much and when he bluffs, what kind of hands he bets, how aggressive he is, etc. One of the most known player classification is the Slansky classification [11].

One important characteristic used for player classification is the percentage of hands he plays. A player can be classified as **tight** if it plays 28% or less hands, and **loose** if he plays more than 28% of the times [43].

Another important characteristic is aggression factor (AF). Aggression factor is given by the following formula (3.3).

$$AF = \frac{(num\_bets + num\_raises)}{(num\_calls)} \tag{3.3}$$

This formula allows classifying players as aggressive, if AF is greater than 1, or passive, if AF is bellow 1.

Aggression factor combined with percentage of played hands leads to four playing styles:

- Loose Aggressive
- Loose Passive
- Tight Passive
- Tight Aggressive

The following figure shows the desired behaviour to be a good player.

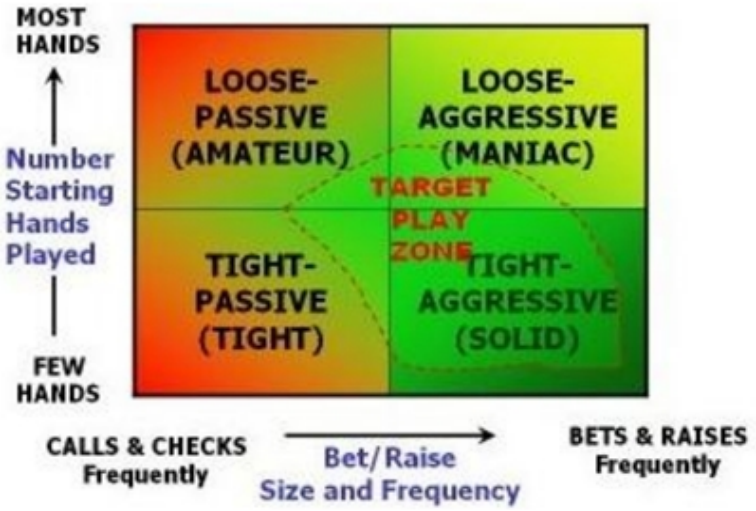

Fig 3.7: Player classification (based on [43])

3.6 Hand Evaluation

The most critical subroutine needed for poker AI is a *Hand Evaluator*. A Hand Evaluator takes a poker hand and maps it to a unique integer rank such that any hand of equal rank is a tie, and any hand of lesser rank loses. A hand evaluator may also classify the hand into a category (a Full House, Tens over Aces). For poker AI, it is absolutely critical to have the fastest hand evaluators possible [49]. Any poker agent may evaluate thousands of hands for each action it will take. It may enumerate all possible hands or by simulating thousands of hands using Monte Carlo Method. These calculations must be really fast, so a poker agent can make a decision in a reasonable amount of time.

3.6.1 Hand Evaluators

A poker hand evaluator is a function that receives a set of cards (5 to 7) and returns a number that means the relative value of that hand. Many hand evaluators were created, with different advantages and disadvantages. Next it will be described the most popular/important hand evaluators to the date.

3.6.1.1 Marv742's Multi Player Flop Tables

MARV742 tables are a database which contains approximations of the probability of winning for any given hand and specific flop, against varying number of opponents [41]. This means that by using these tables a player can quickly identify his odds of winning at any instant of the game.

The main limitation of using MARV742 tables is that they only provide probability to Flop round.

3.6.1.2 Cactus Kev's 5-Card Evaluator

The Cactus Kev's 5-Card Evaluator uses one of fastest hand evaluator algorithms. The idea behind the algorithm is using a pre-computed hand ranking table. The number of possible combinations can be calculated as in (3.4).

$$N = \frac{(n!)}{(r!(n-r)!)}, n = 52 \wedge r = 5 \rightarrow N = \frac{(52!)}{(5!(47!))} \rightarrow N = 2598960 \quad (3.4)$$

Since the number of combinations is not that high, it is quite computationally feasible to store all hands value in a 10mb table (2598960 * 4 bytes). The problem is that all hands must be in order. Cactus Kev's 5 card evaluator uses prime numbers to order the cards, because if you multiply the prime values of the rank of each card in your hand, you get a unique product, regardless of the order of the five cards [50].

Cactus Kev's evaluates cards with the following prime values.

Table 3.4: Cactus Kev's card values

Card	Two	Three	Four	Five	Six	Seven	Eight	Nine	Ten	Jack	Queen	King	Ace
Value	2	3	5	7	11	13	17	19	23	29	31	37	41

For instance a King High Straight hand will always generate a product value of 14,535,931. Since multiplication is one of the fastest calculations a computer can make, we have shaved hundreds of milliseconds off our time had we been forced to sort each hand before evaluation [50].

The only big limitation of this hand evaluator is that it can only be used to evaluate 5-card hands. This means that to use in game variations like Texas Hold'em which needs to evaluate 7-card hands, we had to evaluate all possible 21 combinations of 5 cards to determine which one was the best.

3.6.1.3 Pokersource Poker-Eval

Poker-eval is a C library to evaluate poker hands. The result of the evaluation for a given hand is a number. The general idea is that if the evaluation of your hand is lower than the evaluation of the hand of your opponent, you lose. Many poker variants are supported (draw, Hold'em, Omaha, etc.) and more can be added. poker-eval is designed for speed so that it can be used within poker simulation software using either exhaustive exploration or Monte Carlo [51, 52].

Poker-eval is probably the most used hand evaluator, because of its multiportability of poker variants and its speed of evaluation. The only limitation might be the complexity of its low level API, however there are some third party classes that encapsulate the usage of Poker-Eval API, making it simpler to use.

3.6.1.4 Paul Senzee's 7-Card Evaluator

Paul Senzee's 7 Card Evaluator [51, 53] uses a pre computed hand table to quickly determine the integer value of a given 7 card hand. Each hand is represented by a 52 bit number, where each bit represents an activated card. The total number of activated bits is 7, representing a 7 card hand.

If we had unlimited memory, we could just use the number produced by a given hand to index into an enormous and very sparse array. However, this would require an array with 2^52 entries which means it would not be nowadays computationally feasible as it would require about 9 petabytes of memory (9 million gigabytes).

To solve the problem, Paul Senzee's developed a hash function that turns the hand value into an index between 0 and roughly 133 million and computed a 266mb which is by far a much smaller table.

The limitation of this hand evaluator is that it only evaluates 7 card poker hands, therefore is not portable to other poker variants. Moreover, the code of this evaluator is not complete as it only provides the hash function. So any user has to make a table creation and table loading functions to be able to use this evaluator.

3.6.1.5 TwoPlusTwo Evaluator

TwoPlusTwo evaluator is another lookup table poker hand evaluator with the size of 32487834 entries with a total size of ~250mb [51]. However TwoPlusTwo Evaluator is extremely fast, probably the fastest hand evaluator there is. To get the value of a given hand, the process is just performing one lookup per card. For instance to get a 7 card hand value the code with be just (admitting HR is the lookup table):

```
int GetHandValue(int[] pCards)
{
      int p = HR[53 + pCards[0]];
      p = HR[p + pCards[1]];
      p = HR[p + pCards[2]];
      p = HR[p + pCards[3]];
      p = HR[p + pCards[4]];
      p = HR[p + pCards[5]];
      p = HR[p + pCards[6]];
      return p;
}
```

Fig 3.8: Calculating hand value using TwoPlusTwo Evaluator

The idea behind the implementation of this extremely fast evaluator is a state machine. Each entry on the table represents a state. The next state is the sum of the value of the card and the value of the state. In the final state, the value represents the hand value.

The limitations of this hand evaluator are that it only supports 5, 6 or 7 card evaluation. However, this is not usually a problem since the most used evaluations are usually 5, 6 or 7 card.

3.6.1.6 Foldem Hand Evaluator

Foldem Hand evaluator [54] is a fast 7 card hand evaluator that is able to evaluate about 4,500,000 hands per second. It is far slower than TwoPlusTwo evaluator that is capable of evaluating about 15,000,000 hands per second. Even so it has some advantages over other evaluators. First, it runs with low memory (2mb), which means that it can be used on less equipped computers. Furthermore, it's one of few that was originally written in Java, which means that can save code porting time for Java programmers.

3.6.2 Odds Calculation

Evaluating a hand, which was discussed above, is about giving a score to a set of cards. A Poker AI in reality does not directly use the hand evaluators, because it does not know the opponent's cards, so there is no group of hands to compare.

A Poker AI performs odds calculation, which consists on the prediction of the its own hand success. For that purpose, it evaluates its own hand and compares with possible opponent’s hands. This prediction is used to help measuring the risk of an action.

There are various ways to determine the odds which will be discussed bellow.

3.6.2.1 Chen Formula

Chen Formula is a mathematical formula developed by the professional poker player William Chen [55]. This formula can determine the relative value of a 2 card hand (pocket hand). The steps to determine the value of the hand are:

- Ace = 10 points
- King = 8 points
- Queen = 7 points
- Jack = 6 points
- 10 through 2 = half of face value (i.e. 10 = 5, 9 = 4.5)
- Pairs, multiply score by 2 (i.e. KK = 16), minimum score for a pair is 5 (so pairs of 2 through 4 get a 5 score)
- Suited cards, add two points to highest card score
- Connectors add 1 point (i.e. KQ)
- One gap, subtract 1 point (i.e. T8)
- Two gap, subtract 2 points (i.e. AJ)
- Three gap, subtract 4 points (i.e. J7)
- Four or more gap, subtract 5 points (i.e. A4)

For instance the values of AA, 98suited and K9suited are respectively 20, 7.5 and 6 points. The formula can also be used to get a suggestion of how to play in PreFlop:

- Early Position
  - Raise = Score is 9 or higher
  - Call = Score is 8 or higher
  - Fold = Score is lower than 8
- Middle Position
  - Raise = Score is 9 or higher
  - Call = Score is 7 or higher
  - Fold = Score is lower than 7

- Late Position
  - Raise = Score is 9 or higher
  - Call = Score is 6 or higher
  - Fold = Score is lower than 6

3.6.2.2 Effective Hand Odds

The effective odd of a hand is the probability of a given hand, against a given number of opponents, winning the pot. This means that to calculate the hand odds we need to enumerate all possible card combinations to the board and opponents using the remaining cards, being the remaining cards the ones that are not on the player's hand or on board. This estimate assumes that all opponents will play the hand to the end, without folding.

The problem with this estimate is that the computation of all possible hands is a lengthy process, because of the number of possibilities. One way to solve this problem is to use the Monte Carlo Method. With Monte Carlo Method, a fixed number of random experiences are generated to get an approximation of the hand odds.

The following table was calculated using Monte Carlo Analysis. It presents the calculation of hand odds, using Monte Carlo Method with different number of trials. It is also presented the difference between the exact answer and the estimate answer. It is also presented the time needed to get that estime [56].

Table 3.5: Monte Carlo Analysis to determine effective hand odds [56]

Trials	*Wins*	*Difference*	*Time (s)*
10	80.00%	0.12955	0.0082
50	72.00%	0.04955	0.0001
100	61.00%	0.06045	0.0002
500	66.90%	0.00145	0.0006
1000	65.65%	0.01395	0.0013
5000	66.88%	0.00165	0.0070
10000	67.26%	0.00210	0.0125
15000	66.78%	0.00261	0.0138
20000	67.17%	0.00128	0.0145
25000	66.78%	0.00267	0.0178
30000	66.91%	0.00131	0.0217
40000	67.31%	0.00268	0.0323
50000	66.97%	0.00079	0.0357
100000	66.98%	0.00064	0.0719
1000000	67.04%	0.00009	0.7736
All trials	67.04%	0	299

As it can be seen, it is possible to get very good approximations with much less time.

3.6.2.3 Hand Strength

Hand strength calculation is similar to effective hand odds. However hand strength does not consider the number of opponents and only considers the current board cards (if any). This means that it is much faster to determine the hand strength. To calculate the hand strength we can use the code in the bellow figure.

```
HandStrength(ourcards,boardcards)
{
   ahead = tied = behind = 0
   ourrank = Rank(ourcards,boardcards)
   for-each(oppcards){//all possible opp two card combination
      opprank = Rank(oppcards,boardcards)
      if(ourrank>opprank) ahead++
      else if(ourrank==opprank) tied++
      else behind++
   }

   handstrength = (ahead+tied/2) / (ahead+tied+behind)
   return(handstrength)
}
```

Fig 3.9: Hand Strength

There is also a variation of this algorithm [43] that instead of iterating using all possible two card combinations of the opponents, it only uses the cards that the opponent probably has based on his Slansky's classification [11].

```
HandStrength(ourcards,boardcards, player_classification)
{
   ahead = tied = behind = 0
   ourrank = Rank(ourcards,boardcards)
   //Consider all two-card combinations of the remaining cards.
   for-each(oppcards)
   {
      if(oppcards belong to player_starting_hands_range) {
         opprank = Rank(oppcards,boardcards)
         if(ourrank>opprank) ahead += 1
         else if(ourrank==opprank) tied += 1
         else behind += 1
      }
   }
   handstrength = (ahead+tied/2) / (ahead+tied+behind)
   return(handstrength)
}
```

Fig 3.10: Hand Strength (modified) [43]

3.6.2.4 Hand Potential

The hand potential is an algorithm that calculates PPOT and NPOT which are respectably the positive potential and the negative potential. The PPOT is the chance that a hand that is not currently the best improves to win at the showdown. The negative potential is the chance that a currently leading hand ends up losing. Therefore, PPOT and NPOT are used to determine the hand's potential allowing to estimate the flow of the game.

PPot and NPot are calculated by enumerating over all possible hole cards for the opponent, like the hand strength calculation, and also over all possible board cards. For all combinations of opponent hands and future cards, we count the number of times the agent hand is behind, but ends up ahead (PPot), and the number of times hand is ahead but ends up behind (Npot) [43].

The algorithm to determine PPOT and NPOT is presented on the following figure.

```
HandPotential(ourcards, boardcards, player_classification)
{
   /*Hand Potential array, each index represents ahead, tied, and
behind.*/
   integer array HP[3][3] /* initialize to 0 */
   integer array HPTotal[3] /* initialize to 0 */
   ourrank = Rank(ourcards, boardcards)
   /*Consider all two-card combinations of the remaining cards for
opponent.*/
   for-each(oppcards) {
      opprank = Rank(oppcards,boardcards)
      if(ourrank>opprank) index = ahead
      else if(ourrank=opprank) index = tied
      else index = behind
      HPTotal[index] += 1
      /* All possible board cards to come. */
      for-each(turn) {
         for-each(river)
         { /* Final 5-card board */
            board = [boardcards,turn,river]
            ourbest = Rank(ourcards,board)
            oppbest = Rank(oppcards,board)

            if(ourbest>oppbest) HP[index][ahead] += 1
            else if(ourbest==oppbest) HP[index][tied] += 1
            else HP[index][behind] += 1
         }
      }
   }
   /* PPot: were behind but moved ahead. */
   PPot = (HP[behind][ahead] + HP[behind][tied]/2
   + HP[tied][ahead]/2) / (HPTotal[behind]+HPTotal[tied]/2)
```

```
    /* NPot: were ahead but fell behind. */
    NPot = (HP[ahead][behind] + HP[tied][behind]/2
    + HP[ahead][tied]/2) / (HPTotal[ahead]+HPTotal[tied]/2)

    return(PPot,NPot)
}
```

Fig 3.11: Hand Potential Calculation

3.6.2.5 Software

There are some freeware and commercial applications that can be used to determine hand odds. One of the most popular commercial applications is Poker Academy that can calculate effective hand odds against a given number of opponents. It is also possible to calculate the odds of winning for any given state of the game, by defining which cards each player has and the board cards [57].

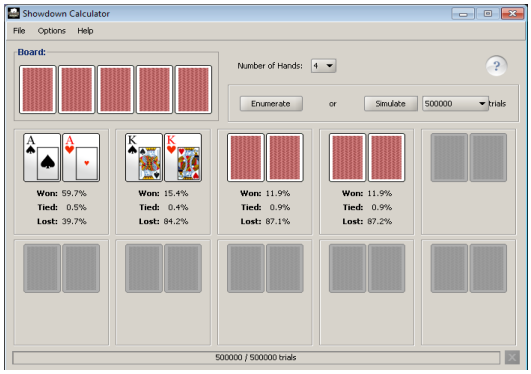

Fig 3.12: Poker Academy Showdown Calculator

Another Software that can be used for this purpose is Poker Stove [58]. This software is very similar to Poker Academy Showdown Calculator, but with the advantage that is free to use.

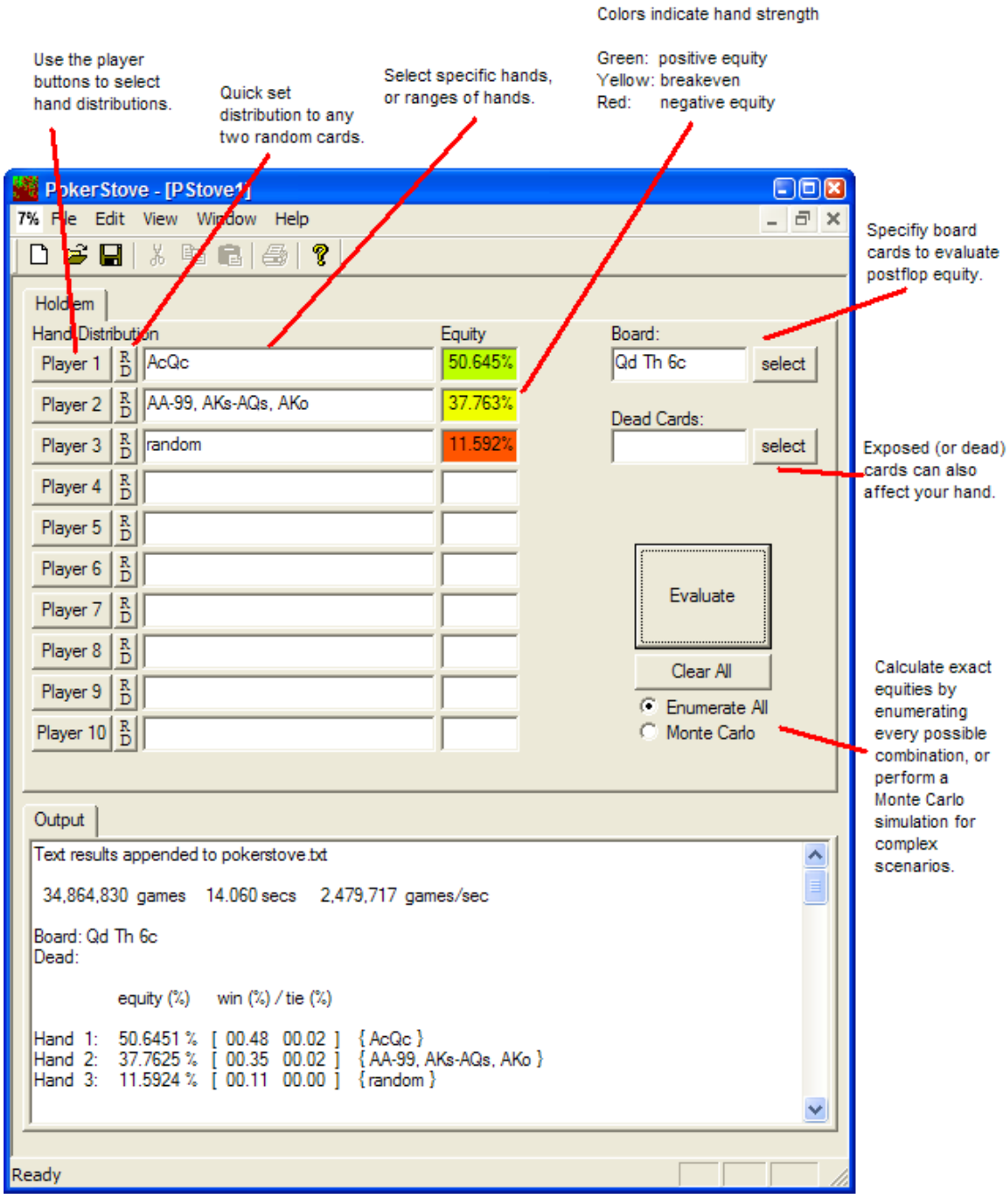

Fig 3.13: Poker Stove [58]

3.7 Data mining

Data mining is the process of getting relevant information, like patterns, associations, anomalies or alterations in a data repository [59]. The main objective of data mining is to extract information that wasn't originally found among the original data set.

There are two typical data mining problems: prevision and description. Prevision problems consist in obtaining estimates for a given knowledge area through analysis of past experiences. Description problems consist in analysing the data set evolution to increase the knowledge about that data set [60]. In this thesis work, the problem to be solved is clearly a prevision problem, because the goal is to predict the next action based on past behaviour thereby mimicking the player strategy.

There are two types of prediction problems: regression and classification [61]. Regression consists in finding relations between data variables [62]. Regarding to classification, this is to define classes of data entries that meet certain patterns [60]. In this thesis, it will be followed the approach of classification where the game table variables are the data entries and the actions are the classes.

The data mining process can be decomposed in four steps [61]:

- Data preparation;
- Data reduction;
- Modelling;
- Solution analysis.

3.7.1 Data preparation

Data preparation is the process of data extraction so that they are in a common format that allows a correct analysis of the problem. This means the definition of the variables and their type. The type of variables might be qualitative (nominal or ordinal) and quantitative (rate or interval). The data is then organized in form of matrix with all attributes and classes. The following table presents an example of a data set.

Table 3.6: Data mining table example

Attribute 1 (nominal) "Person Type" { Sedentary, Sportsman }	*Attribute 2 (rate) "Fat Percentage" Fat weight / Total weight*	*Attribute 3 (interval) "Height" [110-210] cm*	*Class (ordinal) "Evaluation" { Bad, Good, Very Good }*
Sportsman	0.12	180	Very Good
Sedentary	0.30	176	Good
Sedentary	0.74	155	Bad

On the above table we can see examples of the four types of variables. The first attribute "Person Type" is nominal, and can take the values "Sedentary" or "Sportsman". There is no order between the values that "Person Type" can take. Next attribute is "Fat Percentage" that is a rate between fat weight and total body weight. The next attribute is "Height" which can take values between 110 cm and 210 cm (interval type). Finally we have a class "Evaluation" that is

of ordinal type, because there is some order between the possible values as “Very Good” is better than “Good” and “Good” is better than “Bad”. “Evaluation” is also a class, which means that the other attributes determine its value. In this spcific case the goal of data mining classification is to find the function that takes a “Person Type”, “Fat Percentage” and “Height” and returns the body evaluation.

Sometimes it is necessary to normalize data to improve the efficiency of certain classifiers like Artificial Neural Networks [61]. One of the most common attribute normalization is setting the attribute value in a given interval. There are several methods to normalize data: min-max normalization, z-score normalization and normalization by decimal scale [63].

3.7.2 Data reduction

Data reduction is an important step of data mining. The idea of data reduction is the removal of data attributes or data examples in order to improve the classifier efficiency [61]. The first step of data reduction is to analyse the variables and evaluate their prevision potential.

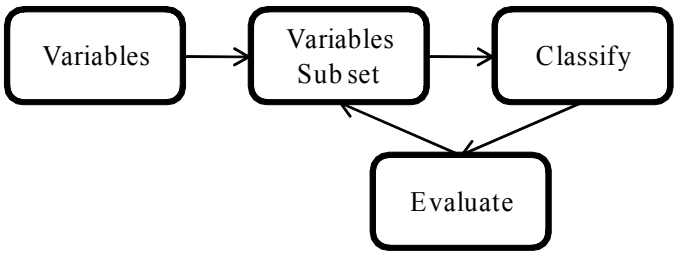

Fig 3.14: Data reduction process

The reduction of the number of variables is an iterative process. First we choose a sub set of variables, build the prediction model and finally evaluate it. After testing with different sub sets of variables, when the prediction model error is reduced to a level set as acceptable, the variables to use in the prediction model were found.

After reducing the number of variables, the next step is to reduce the number of table entries, if the data set is too big. There are various ways to reduce the number of entries like random sampling, stratified sampling and sampling clusters.

3.7.3 Modelling

After preparing and reducing the data, the next step is to obtain the model that represents the association between attributes and classes.

There are various ways/algorithms to create the model. In this thesis makes us of supervised learning classifiers.

Supervised learning is a machine learning technique for deducing a function from training data where the attributes are the function inputs and the class is the function output. This function represents the model of the classifier. Currently there are several supervised learning classifiers. The most important are described in the next subsections.

3.7.3.1 Artificial Neural Networks

Artificial Neural Network (ANN) is a mathematical/computational model that attempts to simulate the structure of biological neural systems. It consists of an interconnected group of artificial neurons and processes information using a connectionist approach. 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.

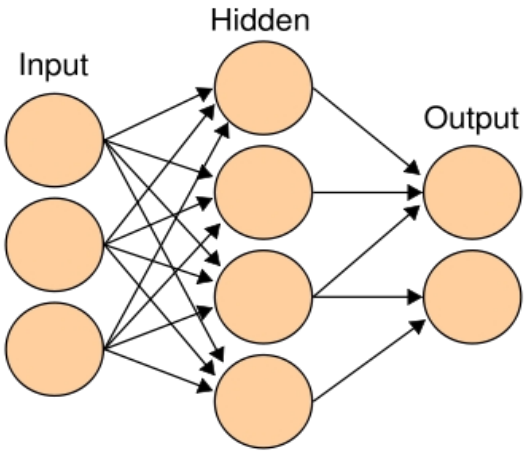

Fig 3.15: Neural Network Example

The above figure represents a simple neural network with 3 layers with each neuron of each layer connected to the neurons of the previous layer and the neurons of the following layer.

The neurons are identical units that are connected by links. The interconnections are used to send signals from one neuron to the other [66]. Each link has a weight that defines its importance to the model. To create a model using an ANN, the link weights must be adjusted using the training data and a training algorithm.

There are some ANN training algorithms like Back Propagation or Resilient Propagation [65] as well as some common network configurations like Multilayer Perceptrons [67].

The main advantage of neural networks is the fact that they eliminate redundant attributes. The main disadvantage is that they take a lot of time to train.

3.7.3.2 Decision Tree's

A decision tree is a decision support tool that uses a tree data structure of decisions and their outcomes and sometimes the chance of each decision or outcome. Decision trees can be used to classify data. The following picture represents a decision tree example to check if the weather is good to play tennis.

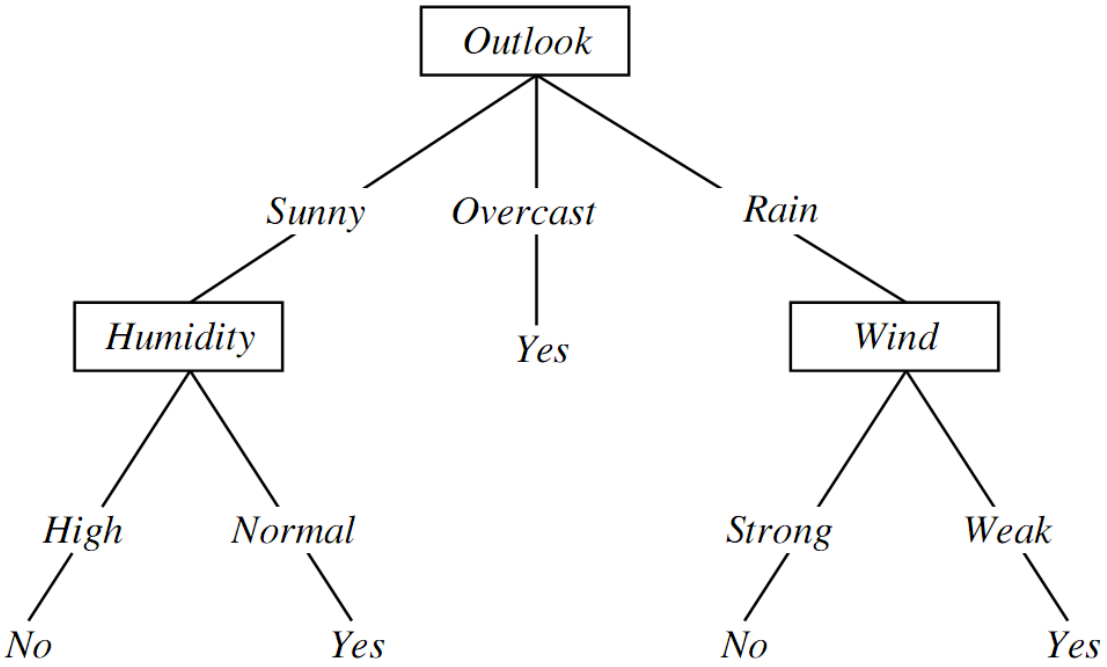

Fig 3.16: Decision tree example

There are various algorithms to create decision trees from training data like C4.5 [71], Random Forests [72] and Best First [73].

The use of decision trees has a lot of advantages over other classifiers, like:

- Simple to understand and interpret;
- Requires little data preparation unlike other techniques which normally require data normalization;
- Able to handle both numerical and categorical data;
- Uses a white box model;
- Possible to validate a model using statistical tests;
- Robust;
- Great efficiency even when the training data is large.

However there are some problems like:

- The problem of creating an optimal decision tree is NP-complete which means that the decision tree creation algorithms won't always generate the best solution.
- Possibility of over fitting i.e. the tree may not generalize the data well.

3.7.3.3 Support Vector Machines

Support vector machines (SVM) are a technique based on statistical learning theory which works very well with high-dimensional data and avoids the curse of dimensionality problem [68]. The objective is to find the optimal separating hyper plane between two classes by maximizing the margin between the classes' closest points. This can be better explained by observation of the following figure.

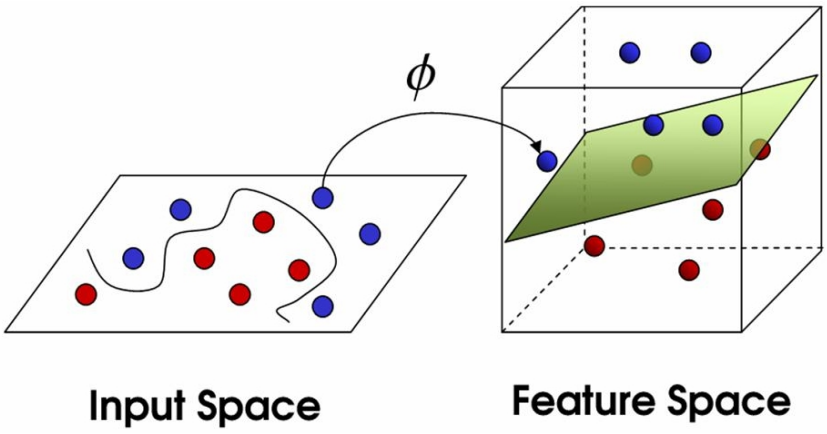

Fig 3.17: Support vector machines

There are clearly two classes on the input space, the blue balls and the red balls. There were separated by the green hyper plane on the feature space. The green plane maximizes the margin between the closest points of the two classes, i.e. the two balls with different colour that were closer.

There are some algorithms based on SVM's that solve multi class classification problems, like Sequential Minimal Optimization [69].

The main advantages of SVMs are [70]:

- SVMs exhibit good generalisation;
- Hypothesis has an explicit dependence on the data (via the support vectors). Hence can readily interpret the model;
- Learning involves optimisation of a convex function (no false minima, unlike a neural network);

- Few parameters required for tuning the learning machine (unlike neural network where the architecture and various parameters must be found);
- Can implement confidence measures.

3.7.4 Solution analysis

There are various techniques to test/estimate the performance of a predictive model. To estimate the performance of a predictive model we can use:

- **Holdout**: divide the training set in two different/equal parts (normally 1/3 and 2/3) and the biggest part is used to train the model and the other to validate it [74].
- **Cross validation**: divide the training set into K sub sets mutually exclusive. Use each subset to train a classifier and the others to test it. The final error is the average error of all validations [75]. Normally the K value is 10, as there is theoretical evidence to choose K=10.
- **Leave-one-out**: divide the training set into K sub sets mutually exclusive. Create a model using all sub sets except one that is going to be used as the test set [74].
- **Resubstitution Method:** the training data is the test data. This model may mislead, because the error rate in practice is most times higher. For this reason, this method isn't much used [74].

To test the performance of a predictive model we can also use the confusion matrix. The confusion matrix shows the number of correct classifications versus the number of predictions done for each class.

3.7.5 Supporting Software

There are several tools in which data mining techniques are implemented and ready to use. Some of these tools even have support for data processing and data normalization.

3.7.5.1 Weka

Waikato Environment for Knowledge Analysis (WEKA) [76] is one of the oldest and one of the most used software for data mining. Weka has implemented the most important open source data mining algorithms and is easy to integrate with custom software, because it provides a complete and easy to use JAVA API. Weka also comes with Weka Explorer which is a user interface which allows to easily use all Weka functionalities. Weka is also free to use.

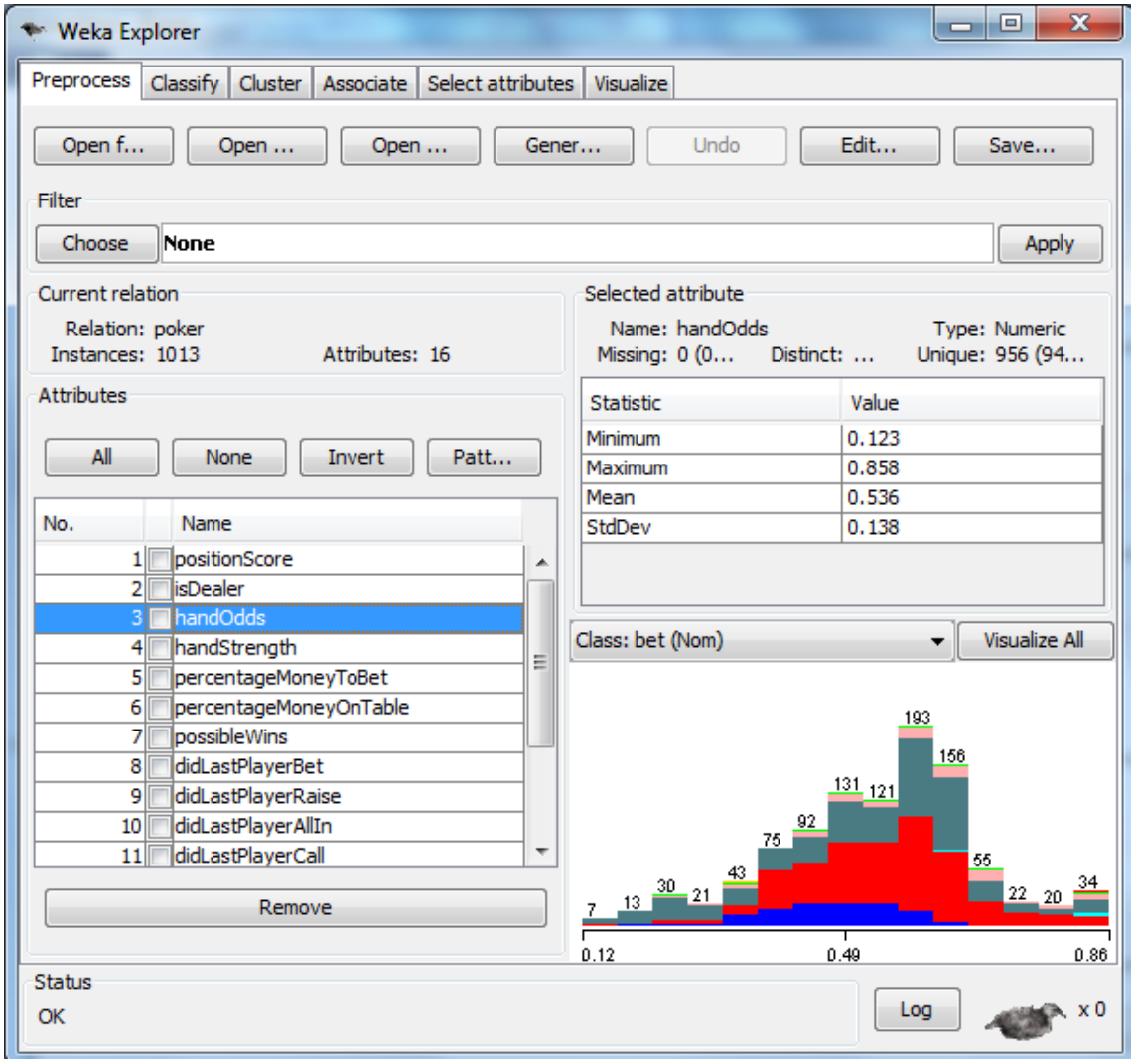

Fig 3.18: Weka explorer user interface

3.7.5.2 Rapid Miner

RapidMiner [77] provides solutions for data mining, text mining and data analysis. The number of operators related with data mining and visual data is high in this package. The tree-based layout with a modular concept also enables breakpoints and re-using building blocks. Another important aspect is concerned with the efficiency because of the layered data view concept, many data transformations are performed at run-time instead of transforming the data and storing the transformed data set. The scalability of RapidMiner has been improving and in the last versions, algorithms were optimized for speed, allowing it to process large amounts of data.

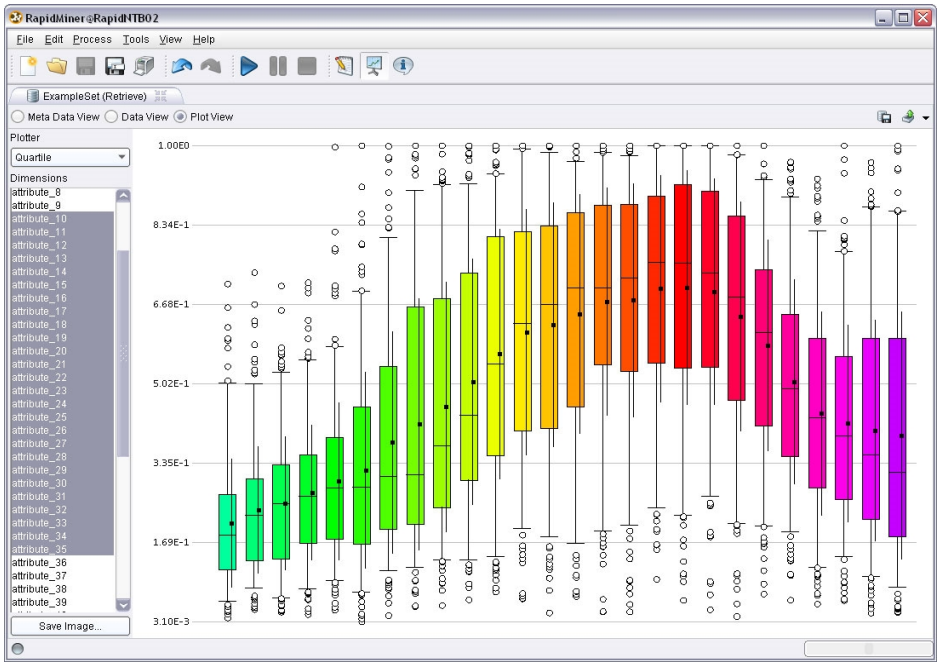

Fig 3.19: RapidMiner user interface

3.8 Poker Agents

A Poker Agent is a piece of software that contains a Poker AI that is used to play Poker autonomously without human interference.

A lot of Poker Agents were developed; in [7] we can find the main architectures of these agents. However, most of poker agents were developed for Fixed Limit Texas Hold'em which is simpler to play than No Limit Texas Hold'em.

3.8.1 Meerkat API

Meerkat API is a commonly used API for testing a Poker AI. Meerkat API is provided by Poker Academy and allows poker bot coders to plug in their own custom bots. This gives an excellent environment for bot development as it is possible to play against any bot developed with Meerkat in a quality GUI or have a custom agent bot play thousands of hands against other bots. Meerkat API also provides easy to use and well documented Java interfaces so anyone can expand them and make its own simulation software [49].

3.8.2 List of Meerkat Api No Limit Texas Hold'em Poker agents

There are a few Meerkat Poker Agents built for No Limit Texas Hold'em. The following table shows the description of some of them.

Table 3.7: List of Meerkat Bots

Name	*Description*
MCTS Bot[78]	The first exploiting no-limit Texas Hold'em bot that can play at a reasonable level in games of more than two players.
Always Call Bot	A very simple and predictable bot that always call every action.
Simple Bot	A bot that uses a simple opponent modelling strategy with hard coded preflop logic and pod odds based decision after the flop.
Hand Strength Bot	A very simple bot that uses a static strategy based on the hand strength. This bot is also capable of bluffing.
Flock Bot	A bot that plays all hand until the River. After the River it sometimes folds.
Chump Bot	A bot that models a very aggressive behaviour. It calls nearly everything, which make it unpredictable.

3.9 Poker agent testing

The final stage of this project is poker agent testing. After generating some strategies from hand history data, they will be used to build poker agents. The first goal of this stage is to verify if strategies were correctly defined. After that, they will be used to support the creation of poker agents.

3.9.1 LIACC's Texas Hold'em Simulator

LIACC Texas Hold’em Simulator is software capable of simulating Texas Hold’em games automatically. It was based on a Server that communicates with Clients through sockets with a predefined communication protocol. The software was developed in C/C++ [43].

This simulation software supports up to 10 clients which could be either human or automated agents. The communications between clients and server is done by TCP/IP protocol. Before starting the simulation, some game options could be defined such as chip stacks, blind value, log file name, etc. The created log file stores information about bets and how much money each player win/loose in each game. Another great feature about this simulator is the communication protocol between agents which is the same as Alberta University Annual Poker Competition [48] so agents tested in this simulator can also be used to participate in this important competition.

Fig 3.20: LIACC's Texas Hold'em Simulator

3.9.2 Poker Academy

One of the best resources for testing a Poker AI is the poker simulation software from Poker Academy [57]. This simulation package contains the Poker AI (Pokibot, Sparbot, Vexbot) developed at the University of Alberta. It was launched in December 2003 as a commercial Poker Training package.

Poker Academy provides a Java based API (named the *Meerkat API*) that allows poker bot coders to plug in their own custom bots. This gives an excellent environment for bot development as you can easily play against your bot in a quality GUI or have your bot play thousands of hands against the Pokibot AI. The program also keeps track of all the hands played and can display comprehensive graphs and analysis of the player statistics.

One of the problems of Poker Academy is that it is misfit for extensive simulations, because of the heavy user interface that results in low simulation speeds. Other problem is that it is not possible to start a simulation without a human player, which means that in each simulation there will always be an additional ghost player that always folds.

3.9.3 AAAI Poker Server

The AAAI Poker Server is an application made to simulate thousands of games between poker agents. This application is used to determine the winner of the Annual Poker Agent Competition organized by University Of Alberta [48].

The agents connect to the server and communicate with it using a TCP/IP protocol. The application is used for 3 different competitions:

- Two heads-up limit competitions
- Two heads-up no limit competitions
- Two 3-player ring limit competitions

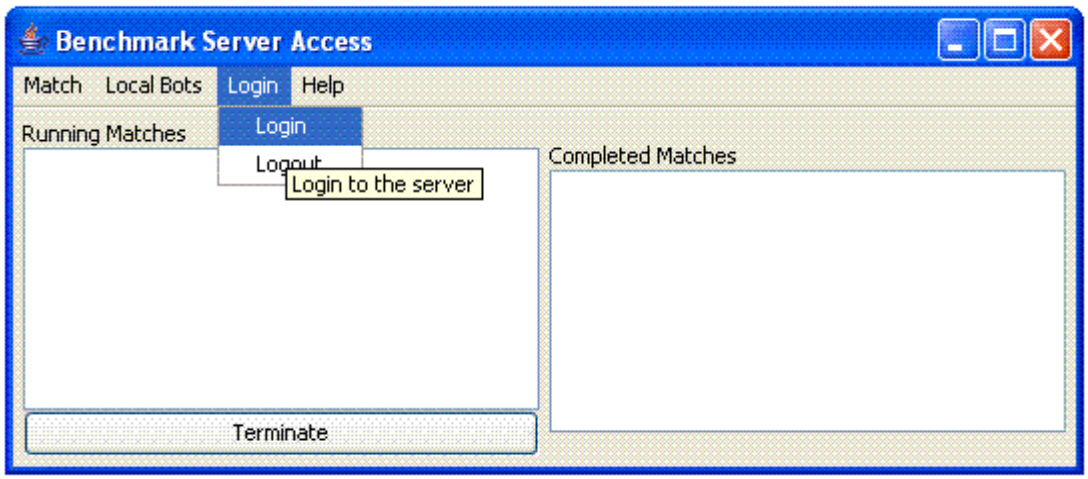

Fig 3.21: AAAI Poker Server [48]

3.9.4 Open Meerkat Poker Testbed

Open Meerkat Poker Testbed [79] is an open source implementation of the Meerkat API for running poker games. It imitates the Poker Academy simulator; however it is much faster because it lacks a heavy user interface.

This application supports Fixed/No-Limit cash games with automatic rebuy. It generates bankroll evolution plots, implements seat permutation (replay games with same cards but with different seat order) and generates game logs. It also shows online bankroll evolution graph.

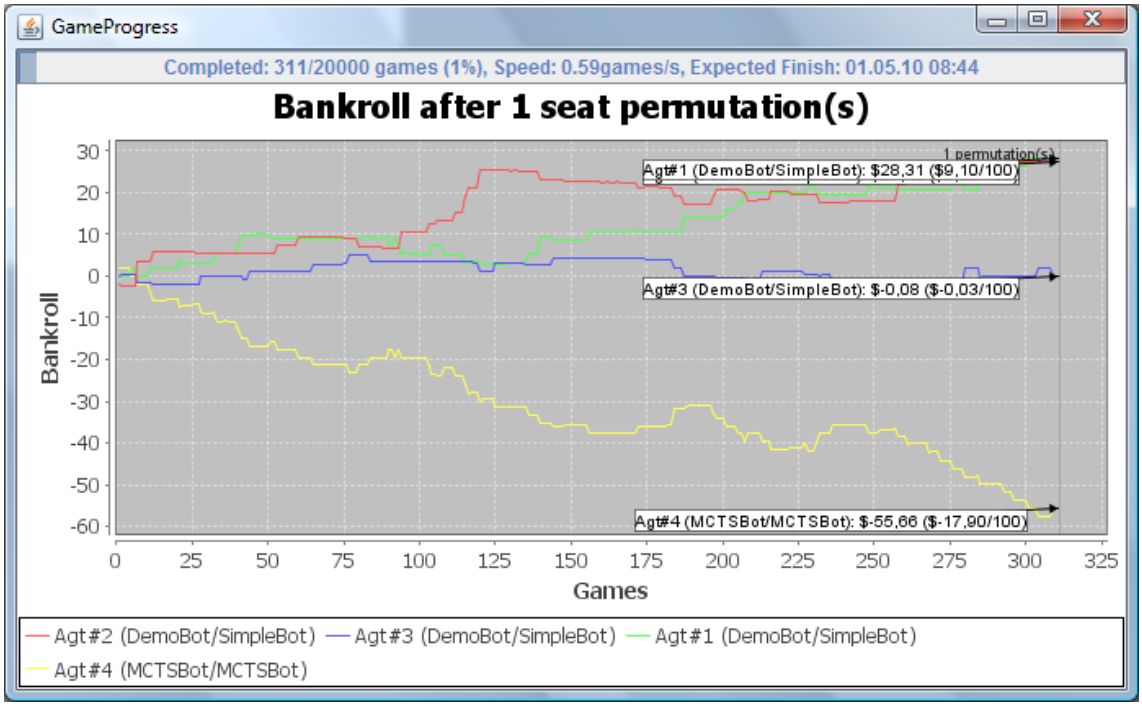

Fig 3.22: Open Meerkat Poker Testbed

3.9.5 Poker Bots

A bot is an automated application that runs automated tasks. These tasks are usually simple but repetitive. Bots perform these tasks at a much higher rate than would be possible for a human alone [45].

A poker bot is a poker agent that plays fully automated in online casinos. Most online casinos disallow their usage because they give an unfair advantage, so its usage is considered cheating. Using a poker bot normally results in being banned from the casino.

Despite poker bot usage being forbidden, they represent the best theoretical way to really test an agent. Playing against human players is best way to test the agent abilities.

There are many poker bot applications available however most are commercial applications.

3.9.5.1 Shanky Holdem Poker Bot

Shanky Holdem Poker Bot [46] is a commercial bot that will play limit and no limit Texas Hold'em in some online casinos, most notably at Full Tilt Poker [33].

This bot was one of the first that played no limit version of Texas Hold'em being [47]. Besides that it has some nice features like running in hide mode (casino clients can't find its process running), random button pixel clicking, some pre-recorded sentences for auto-chatting, bathroom breaks, etc. These characteristics are fully customizable.

It's also possible to customize bot strategy. It's unknown if the defined strategy uses some advanced knowledge like opponent modelling. Even so, it's possible to totally define the bot strategy by using Poker Programming Language.

Fig 3.23: Shanky Technologies Hold'em Poker Bot

3.9.5.2 Win Holdem

WinHoldEm [81] is a commercial poker bot created by Ray Bornert. WinHoldEm was one of the first programmable poker bot frameworks. Users can develop and compile standard C/C++ WHUSER.dll and by this method incorporate his or her own Artificial Intelligence (AI) logic to the bot.

WinHoldEm is the only publicly available pokerbot framework that openly admits support for collusion (which is a type of cheating where the players teamwork to get an unfair advantage) between bots playing on a single table.

3.9.5.3 Open Holdem

OpenHoldem [80] is an open source screen scraping framework and programmable logic engine for the online Texas Hold'em poker game. This framework is similar to WinHoldEm but it has the advantage of being free.

This framework does not include automated collusion capabilities.

It does include:

- A parameter driven engine for screen scraping and interpreting game states (Table Maps)
- A logic engine for making poker decisions based on the game states
- A simplistic scripting language for describing how these poker decisions should be made (using the Spirit parser library)

- Various interface mechanisms allowing for the creation of decision logic via alternative means (C++, Perl, etc)
- An engine for applying the poker decision to the casino table (Autoplayer)

3.10 Summary

In this chapter it was described some research and tools that supported this thesis work. The choice of content for each subject was focused in relevant information needed for this type of research work, instead of general poker research information.

4 Poker agent specification

In this chapter it is presented the developed agent specification, including the global agent architecture and all the necessary development steps to implement it. The chapter also specifies the test methodology that was used to validate the proposed architecture.

4.1 Overview

The goal of this thesis is to create an agent based on game logs by copying the tactics of good players present on them. It is intended to determine whether it is possible to create an agent from scratch by observing how the best human players play. Another goal is to determine if the generated agent is capable of being competitive.

A tactic is a set of rules that determines which action will be taken by a given player in a certain state of the game. The strategy that will be followed by the generated agents will be composed of human player tactics. The way the agent use the learned tactics will be discussed later.

The creation of the agent will be managed by a framework called HoldemML. The HoldemML framework get as input a set of game logs and from them and it will create many different strategies that will be used by HoldeMLBot agent as specified in the figure 4.1.

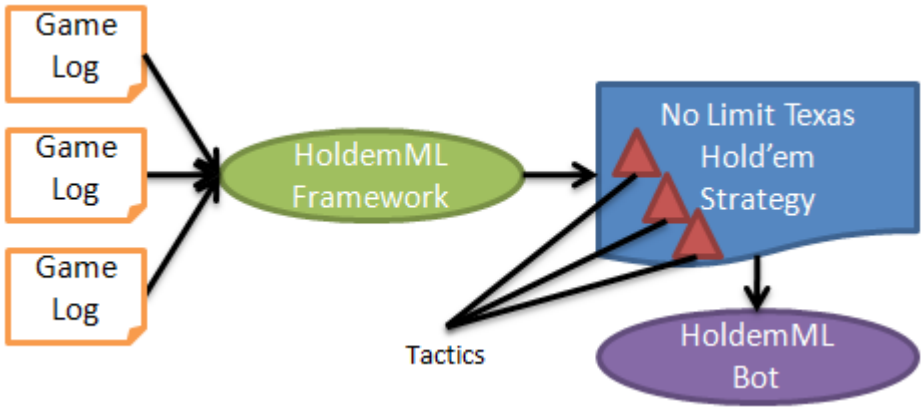

Fig 4.1: HoldemML strategy generation

4.2 HoldemML framework architecture

The HoldemML framework global architecture can be analysed in figure 4.2. Initially we have different data sources that contain poker game data. Since each data source can represent the data differently, we must convert the data into a common format in order to combine the information, because the same player can play in different online casinos, therefore the player's data is scattered in different data sources. The role of converting data into a common format is played by HoldemML Converter. It is also possible to verify whether the documents from data sources are already in the format set as default, using the HoldemML Validator module.

Having the data ready to process, the next step is to extract the game variables that will be used to teach the agents to play. The module that extracts the game variables is HoldemML Stats Extractor, generating a game stats file. HoldemML Stats Extractor requires as input the games logs and a player list. The player list is created by HoldemML Player List Extractor and it presents the list of all players present on the game logs as well as some relevant information like the number of games that each player participated. This information is used as helper to generate the stats file. Using the game participation count information, the HoldemML Stats Extractor is able to only extract the stats of players that have reasonable amount of information.

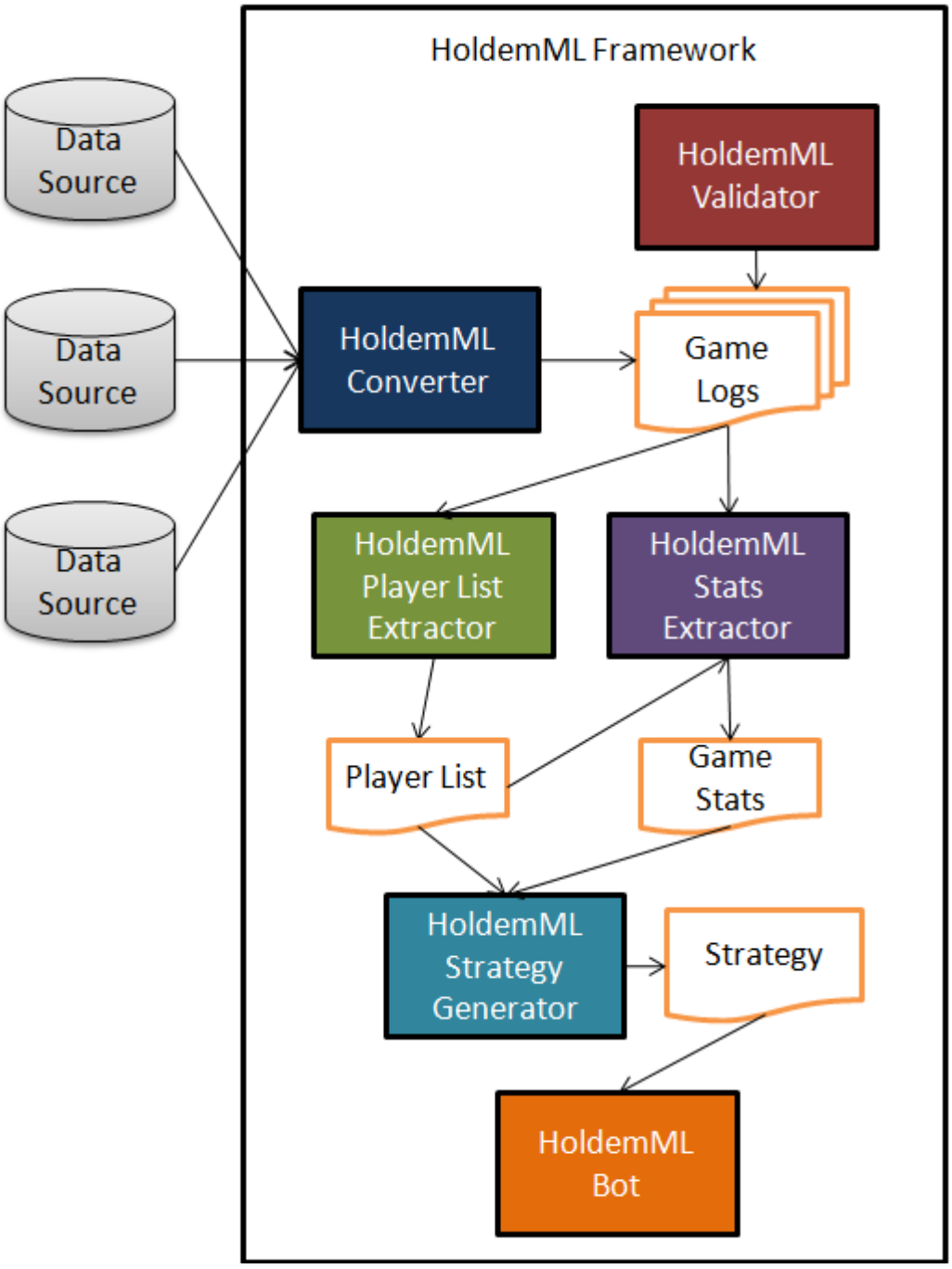

Fig 4.2: HoldemML Framework architecture

After generating the game stats file and the player list file, they are used by HoldemML Strategy Generator to create a strategy file. The game stats file is the main source of information to generate a strategy, but the player list is also useful to give information about opponents. The way the HoldemML Strategy Generator uses the player list will be explained later.

Finally, after generating a strategy file, it can be used by a HoldemML Bot to play poker. The HoldemML Bot module is a Meerkat [49] no limit poker bot that is able to follow the specification of any given strategy file.

4.3 Development phases

In order to implement the architecture defined for the framework and the agent, the development process was divided into the following steps:

- **Data extraction** – extraction of game logs, from different data sources, to a common format.
- **Data analysis –** extraction of game state variables from the players that have a reasonable amount of data.
- **Classifiers training** – train different classifiers to identify what action should be taken for a given game state, using the extracted game state variables.
- **Classifiers evaluation** – evaluate the classifiers that were trained in order to determine which one is the most suitable to recognize which action should be taken for a given game state.
- **Agent building –** create an agent, using Meerkat API [49] that uses the classifiers to determine its actions.
- **Agent testing** - evaluate the behaviour of the agent and verify that it is similar to the behaviour of the original player. Test the agent against other agents and analyze the results.

4.3.1 Data extraction

In data extraction phase, there were implemented the modules that permit the extraction of game history from casino client logs, namely Fult Tilt Poker and Poker Stars. Since there is no standard to represent game history, it was defined a common format to represent the extracted hands.

Two applications that use the HoldemML Framework were create in this phase: one that allows for the conversion of casino logs to the defined common format, and one that allows documents validation in that format.

4.3.2 Data analysis

In data analysis phase, first the game variables that were considered relevant to influence the players' actions were defined. Secondly, the modules that permit the extraction of these game variables to files were implemented.

Two applications that use HoldemML Framework were also created on this phase: one that allows the extraction of a player list from game logs and another that extracts lists of game variables where each list belongs to a player. Only players with a certain minimum of plays were extracted.

4.3.3 Training and evaluating classifiers

After extracting game variables, the lists were used along with Weka to train various different classifiers. The classifiers were evaluated and it was determined which one was more suitable to this domain. There were also implemented the training modules of the HoldemML framework.

An application that uses HoldemML Framework was also created on this phase: one that allows the generation of game strategies from game variable lists.

4.3.4 Agent building

In agent building phase, it was implemented an agent that follows any strategy file created by the HoldemML Framework. The agent was implemented using the Meerkat API [49].

4.3.5 Agent testing

After building the Meerkat agent it was developed an application based on Meerkat Open Test Bed to test the agent with different strategies. Various types of testing were applied:

- **Behaviour testing**: test the agent behaviour to see if it matches the real player behaviour.
- **Inner testing:** test the agent in games with only HoldemML agents. This type of testing is used to analyse if a tactic generated from the best theoretical players (those who earned more money) can win against a tactic generated from the worst players.
- **Outer testing**: test if the agents produced by HoldemML Framework can be competitive against other agents.

4.4 Summary

This chapter briefly explained the process of building and testing the agent and the framework developed, as well as its architecture and the overall operating process.

5 Poker game data extraction

This chapter presents the data extraction/analysis development stages in order to define the agent's strategy as well as the HoldemML Framework applications created for that outcome.

5.1 Overview

The first step to create the specified agent is to obtain poker game data. There are several data sources that can be used, but the chosen one was game logs saved by casino clients. After searching the web, a package of 7,5 GB of game logs was found [84, 85].

There are some prerequisites that the data source must be accomplished in order to be possible to retrieve strategies. The prerequisites are:

- The data source must contain hands where the players show their pocket cards. It is impossible to know the player's strategy without knowing his cards.
- The number of hands shown must be significant. The more hands we have from a certain player, the more chance we have to learn the tactics correctly.
- The data source should contain hands of good players, that is, hands of players that won lots of money. If we learn tactics from good players, we have greater probability of defining a better agent strategy.
- The data source should contain hands of games between human players.
- If possible, the data source should contain the players' cards even when they fold the hand, so it's possible to teach the agent how to fold.

The used data source met all prerequisites, with the exception of the latter one. The latter prerequisite is very difficult to fulfil, since the majority of available data sources on the web are from player's perspective. It would only be possible to obtain data in these conditions through

the online casinos, by monitoring the games. Moreover, access to such data sources is typically paid.

Table 5.1 presents some characteristics of the data source used.

Table 5.1: Used game logs characteristics

Total Size	7,51 Gb (zip compressed)
Total number of games	51.377.820
Number of players	158.035
Total number of showdowns	2.323.538
Number Players with more than 100 showdowns	3.764
Number Players with more than 500 showdowns	183
Showdown Ratio	4,52%
Total number of shows of players with more than 500 showdowns	139.191
Average action count of players with more than 500 showdowns	2.605

As it can be seen from the table, the percentage of games in which occurs exposing of the cards is only 4.5%. Similarly, the number of players that show their cards is much smaller than the total number of players, representing only 0.12% of them.

A sample was extracted from this data source, which contains the players with more than 500 shows. The sample contains 183 players, with a total of 139.191 games, and each player has an average of 2.605 actions to analyze.

5.2 HoldemML Format

The used game log package has logs from 6 different casinos, making it necessary to create a common format to represent game history data. After defining a common format, all documents must be converted into that format.

The technology that was chosen to create this file format was XML. The usage of XML has a lot of advantages, like portability, interoperability and supporting tools. Thus, by creating an XML format to represent the hands of poker, we can easily process the information contained therein, as well as support data reutilization.

The file format was named HoldemML, and it can be represented by the tree in figure 5.1.

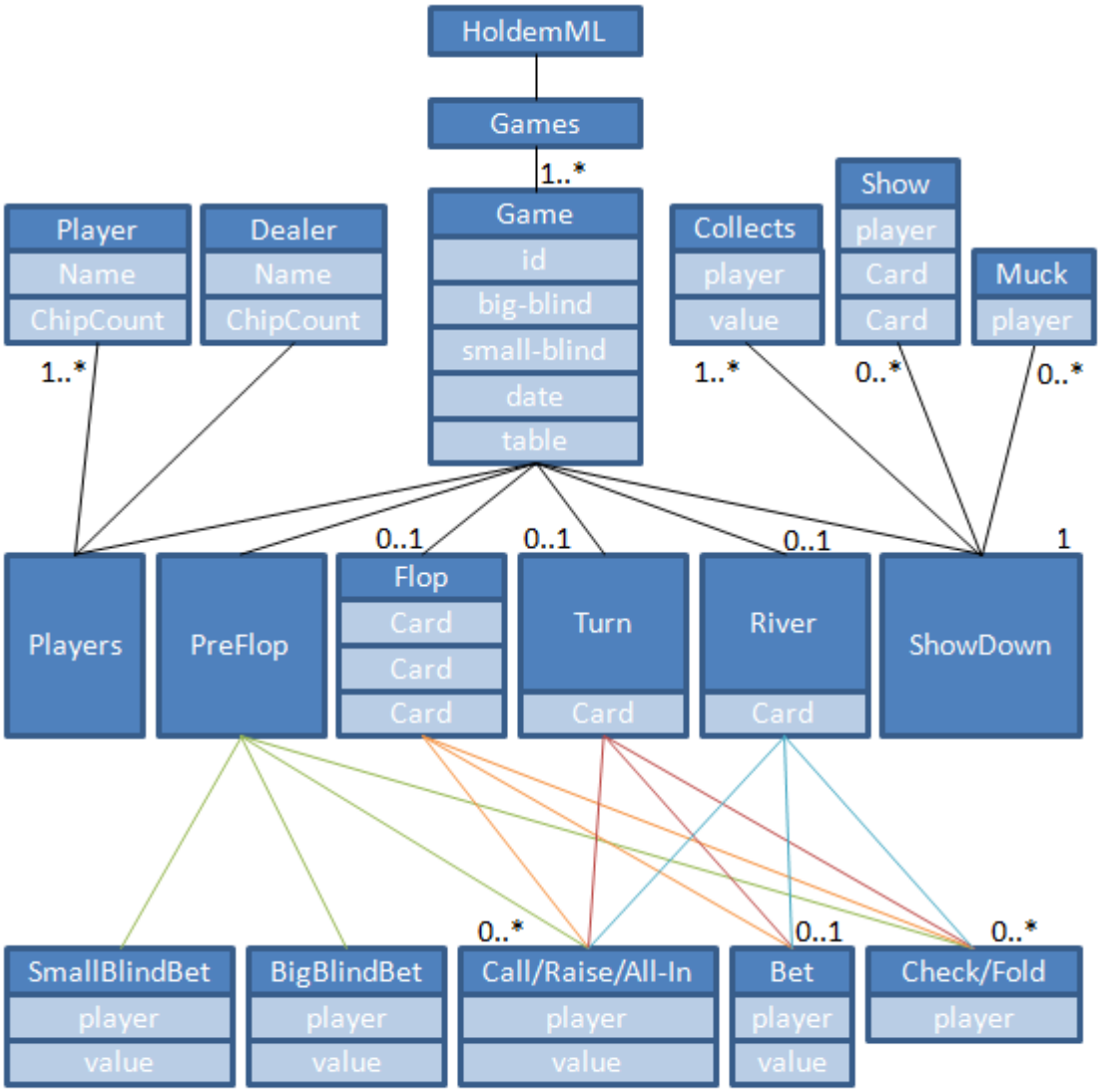

Fig 5.1: HoldemML Schema

Each HoldemML document starts with a <***HoldemML***> element that contains the <***Games***> element. The <***Games***> element represents a set of games, therefore it contains many elements of type <***Game***> (at least 1). Each <***Game***> has attributes associated with it namely id (game unique identifier), big-blind, small-blind, date and table. Each <***Game***> contains the element <***Players***> that contains all the players and their initial bankroll. After the <Players> element, there is an element for each round of the game: <***PreFlop***>, <***Flop***>, <***Turn***> and <***River***>. Each round element contains various actions elements: <***SmallBlindBet***>, <***BigBlindBet***>, <***Call***>, <***Raise***>, <***All-In***>, <***Bet***>, <***Check***> and <***Fold***>. Finally, we have the element <***ShowDown***> in game that represents the round where the players show their cards or collect money. Each element <***ShowDown***> contains at least one element <***Collects***> (the player that won the pot) and may contain elements of type <***Show***> and <***Muck***>.

An example of HoldemML instance can be seen on fig 5.2.

```
<Game id="1" big-blind="6" small-blind="3" date="2009-07-06" table="tableX">
      <Players>
        <Player>
          <Name>Joe</Name>
          <ChipCount>60.1</ChipCount>
        </Player>
        <Dealer>
          <Name>John</Name>
          <ChipCount>318.35</ChipCount>
        </Dealer>
      </Players>
      <PreFlop>
        <SmallBlindBet player="Joe" value="3" />
        <BigBlindBet player="John" value="6" />
        <Call player="Joe" value="3" />
        <Check player="John" />
      </PreFlop>
      <Flop>
        <CommunityCards>
          <Card rank="K" suit="C" />
          <Card rank="6" suit="C" />
          <Card rank="K" suit="H" />
        </CommunityCards>
        <Bet player="Joe" value="18" />
        <Call player="John" value="18" />
      </Flop>
      <Turn>
        <CommunityCards>
          <Card rank="3" suit="D" />
        </CommunityCards>
        <Check player="Joe" />
        <Check player="John" />
      </Turn>
      <River>
        <CommunityCards>
          <Card rank="2" suit="D" />
        </CommunityCards>
        <Check player="Joe" />
        <Check player="John" />
      </River>
      <ShowDown>
        <Show player="John">
          <Card rank="A" suit="C" />
          <Card rank="J" suit="C" />
        </Show>
        <Show player="Joe">
          <Card rank="10" suit="S" />
          <Card rank="A" suit="H" />
        </Show>
        <Collects player="John" value="100" />
      </ShowDown>
    </Game>
```

Fig 5.2: HoldemML document example

5.3 HoldemML Converter

After defining the HoldemML format, the next step is to convert the game logs to that format.

Before converting the game log files, two modules were defined to import and export HoldemML documents, as specified on the figure bellow, in order to support the processing of this type of documents.

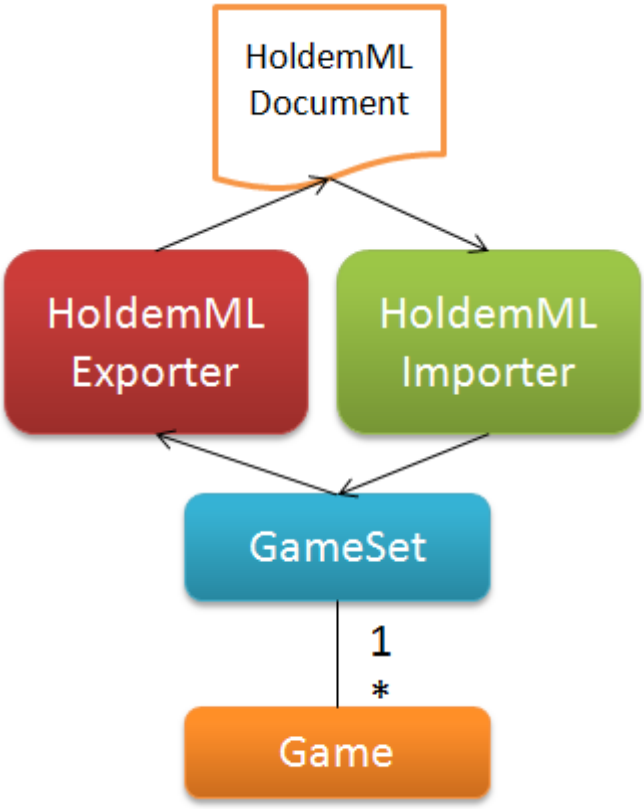

Fig 5.3: HoldemML importer and HoldemML exporter

The diagram on fig 5.4 presents the role of HoldemML Converter in the conversion process. This module receives logs from different sources and returns a Game Set. Using the refereed above exporting module (fig 5.3), it is possible to create a HoldemML file that corresponds to the input data, just by exporting the Game Set using HoldemML Exporter.

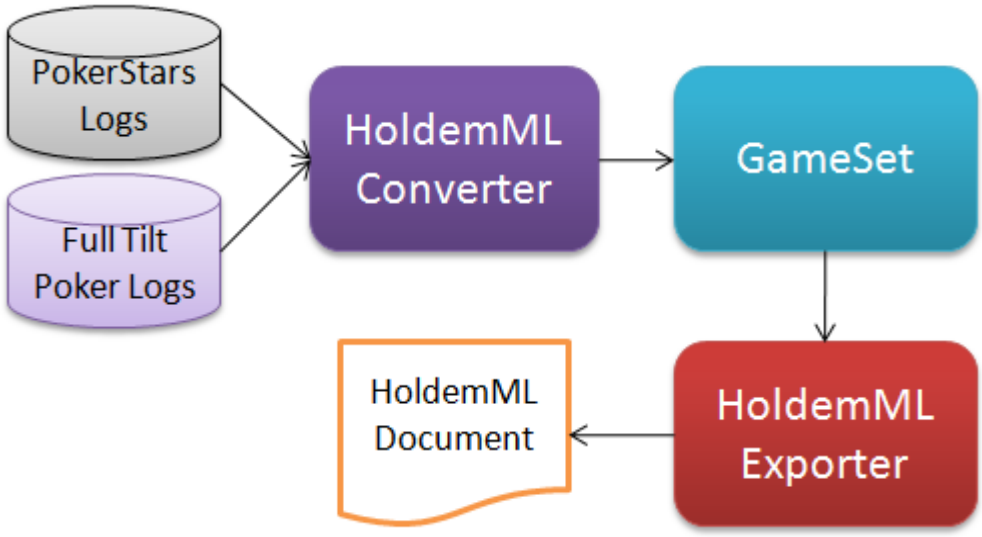

Fig 5.4: HoldemML Converter Role

The process of reading a Game Set from a game log can be represented by the following code.

```
GameSet readGameLog(file)
{
   gameSet = new GameSet();
   file.open();
   while(!file.eof()) {
      line = file.readLine();
      if(line != "") {
         header = readGameHeader(file);
         actions = readGameActions(file);
         if(header != null && actions != null) { //!Check Noise
            game = new Game(header, actions);
            gameSet.addGame(game);
         }
      }
   }
   return gameSet;
}
```

Fig 5.5: Conversion pseudo-code

Should be noted the commented section. The acquired games logs present some noise, like missing values, non existing players and so on. These games were discarded, so the HoldemML file has fewer games than the original files. The number of games discarded games was so little (about 100 of 51 million) that this fact was almost irrelevant.

The produced converter is easily expansible, to support new input formats, throw class inheritance. The only functions that have to be implemented are *readGameHeader* and *readGameActions*.

An application that uses HoldemML Converter was created. The application is very simple to use. The user only has to choose the game logs type, prompt the location of the game logs, the output name and output dir of HoldemML files and hit the convert button.

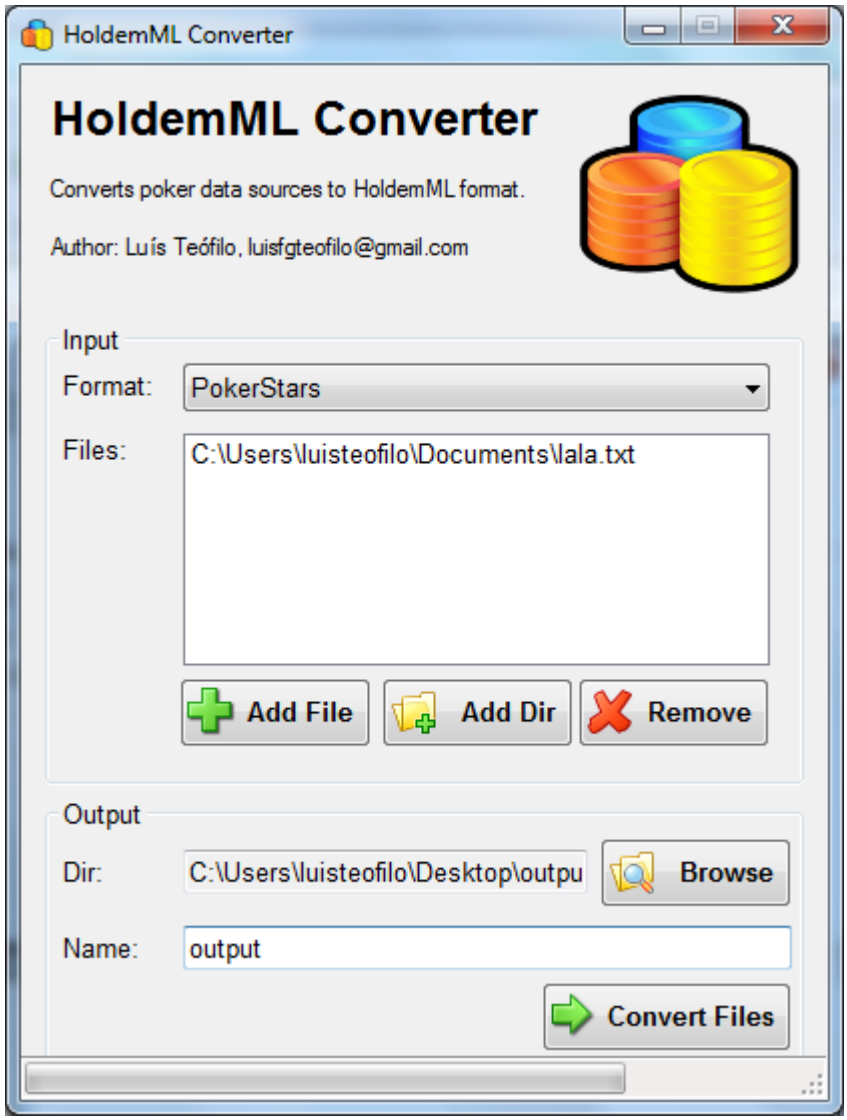

Fig 5.6: HoldemML Converter Application

5.4 HoldemML Validator

Before importing any document to the framework, it should be verified if that document follows the model specified on the HoldemML Schema. For this reason a HoldemML Validator module was created.

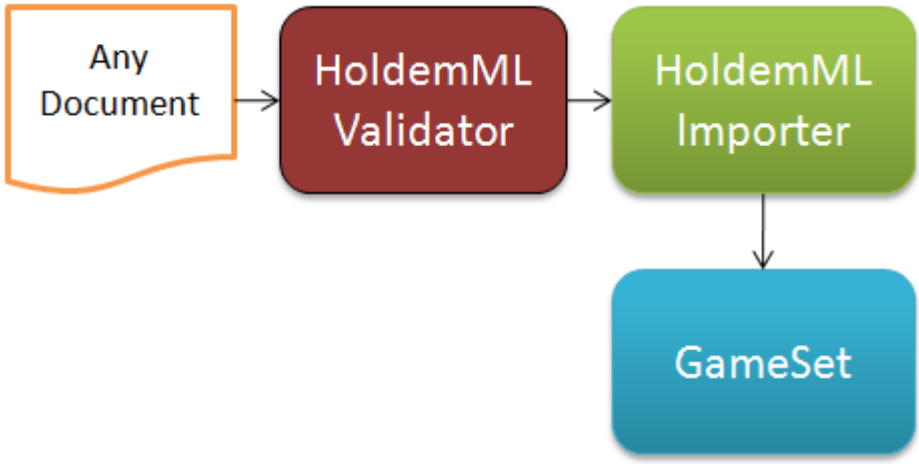

Fig 5.7: HoldemML file importing

An application that uses the HoldemML validation module was also created. This application can be used to check if a given document or set of documents are HoldemML documents (fig 5.8).

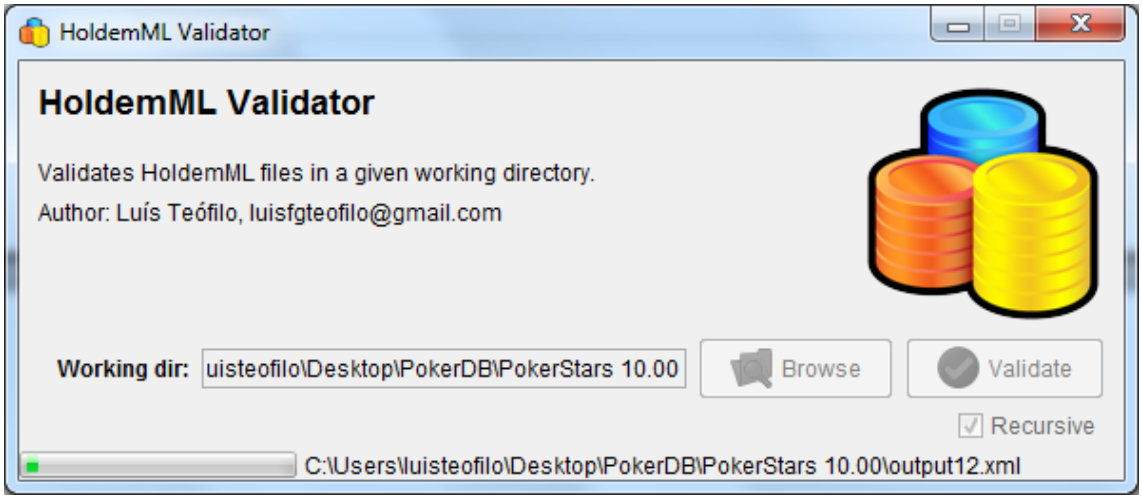

Fig 5.8: HoldemML Validator application

5.5 HoldemML Player List Extractor

The creation of a player list referring to the collected data is an important step for the agent creation. The player list contains the following data:

- Player name: the string that identifies the player;
- Game participation counter: number of games in which the player participates;

- Money Won: total money won by the player;
- Num All-Ins: number of times the player did raise/bet all his chips;
- Num Bets: number of times the player did bet;
- Num Calls: number of times the player did call;
- Num Pre-Flops: number of times the player paid to see the Flop;
- Num Raises: number of times the player did raise;
- Shows Count: number of times the player reach the showdown round;
- Wins Count: number of times the player won the pot;
- Percentage of played hands: percentage of times that the player paid to see the Flop;
- Aggression Factor: Slansky aggression factor;
- Player Type: Slansky player classification.

The extraction of these player parameters mainly serves to support the game variable extraction.

Using game participation counter and money won parameters it is possible to remove players with less participations and negative earnings. Players with few participations can't be used to train classifiers thus its removal speeds up the extraction process of game variables. Players with negative earnings might also be unuseful, because if they lost money, there is the probability that their tactic wasn't good, and it's not consistent to teach bad tactics to the agent. However, it could be useful to get bad player tactics to prove the concept of this thesis. If we create good agents based on good real players and bad agents based on bad real players, it is expected that the good agents will win against the bad agents.

Using the other parameters, we can classify any player present on the games. This could be useful if we choose to extract game state variables that specify the type of opponents that we have on the table.

The process for creating a player list can be represented by the following pseudo-code algorithm.

```
PlayerList createPlayerList(file)
{
   gameSet = HoldemMLImporter(file).import();
   playerList = PlayerList();

   for-each(game in gameSet.games) {
      for-each(player in game.players) {
         playerList.addParticipation(player);
         if(game.flop != null &&
            game.flop.players.contains(player)){//payed preflop
            playerList.addPlayedHand(player);
         }
      }
```

```
        actionUpdate(player, playerlist, game.preflop);
        actionUpdate(player, playerlist, game.flop);
        actionUpdate(player, playerlist, game.turn);
        actionUpdate(player, playerlist, game.river);
    }

    playerList.classifyPlayers();
    return playerList;
}

void actionUpdate(player, playerlist, round)
{
    for-each(action in round.actions) {
        playerList.updateWinnings(player, action.value);
        playerList.updateActionCount(player, action.type);
    }
}
```

Fig 5.9: Conversion pseudo-code

A simple application to extract the player list was created. To use it we just need to indicate the directory where the HoldemML files are located and the output file path. Due to the high number of game logs, the process took about one hour to be completed.

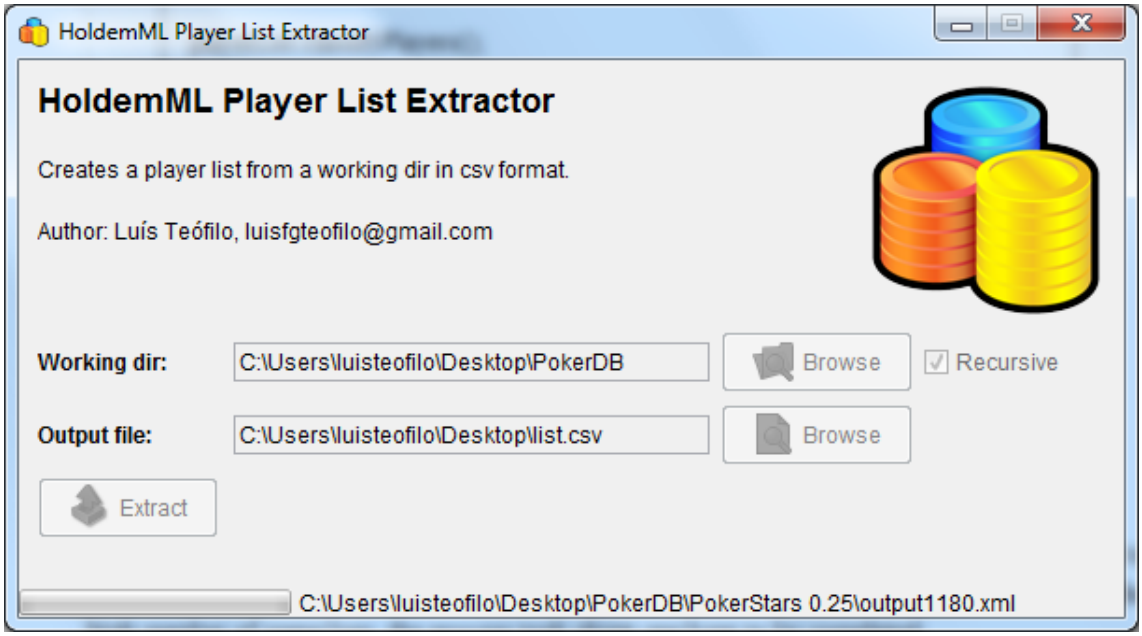

Fig 5.10: HoldemML Player List Extractor

5.6 HoldemML Stats Extractor

After obtaining the player list, it is now possible to extract the game variables from the data files.

The chosen game variables reflect the various aspects of the game. There were divided in two distinct groups: the input group, which define the hypothetical variables that can influence the player decision and the output group, which contains the variables that represent the player decision.

As for the input group, the aspects of the game that were taken in consideration are:

- **Slansky player classification**: the player classification in terms of tightness and aggression. This information is extracted from the player list and is constant for any game state;
- **Position in table**: the position that the player occupies in table can influence the decision. For instance, a player in the beginning positions must act without knowing what his opponents will do next.
- **Player hand:** the quality of the player hand given by the odds.
- **Money:** how much money is involved in the current state.
- **Slansky last opponent classification**: most of the times, the player decision is based on the last opponent's decision, therefore it might be useful to know which type of player is the opponent that acted before the player.
- **Last Opponent action:** as explained in paragraph above, the last opponent's action might influence the player's decision. This represents the last opponent's action.

Considering this aspects of the game, the game variables chosen are shown on table 5.2. All variables type are real, Boolean or enumerations. This prevents wasting time in data preparation, because the generality of the classifiers have a preference for these types of variables.

Table 5.2: Input game variables

Variable	*Type*	*Calculation*	*Description*
Is Aggressive	Boolean	Aggression Factor > 1	The player is considered aggressive in Slansky Classification.
Is Tight	Boolean	% hands played <= 28%	The player is considered tight in Slansky Classification.

PositionScore	Real	(position+1)/numInitialPlayers	How much later is the position.
IsDealer	Boolean	position == 0	The player position is the button.
Hand Odds	Real	Hand Evaluator	Hand effective odds.
Hand Strength	Real	Hand Evaluator	Hand strength.
Percentage Money to Call	Real	callValue / actualMoney	Percentage of chips that the player has to put to match the highest bet.
Percentage Money on Table	Real	moneyOnTable / initialMoney	Percentage of chips that the player has put on the pot in the current game.
Possible Wins	Real	min(1,(pot – moneyOnTable – callValue) / initialMoney)	How much relative money can the player win, after calling the bet.
Table occupation ratio	Real	numberPlayers / numberInitialPlayers	Percentage of players still on the table.
Is last player aggressive	Boolean	Aggression Factor > 1	The opponent is considered aggressive in Slansky Classification.
Is last player tight	Boolean	% hands played <= 28%	The opponent is considered tight in Slansky Classification.
Opponent's action	{Check, Raise, Bet, All-in, Call}	----------	Last action taken by an opponent without being Fold.
Round	{PreFlop, Flop,Turn, River}	----------	Current game round.
Npot	Real	Hand Evaluator	Negative pot potential.
Ppot	Real	Hand Evaluator	Positive pot potential.

For the output variable, we have:

Table 5.3: Output game variables

Variable	*Type*	*Calculation*	*Description*
Percentage of betted money	Real	betValue / actualMoney	Percentage of money bet.
Action	{Check, Raise, Bet, All-in, Call}	----------	Action taken by the player.

It is possible to observe that it's trivial to get most of these variables. However there could be trouble when calculating hand odds.

Calculating hand odds is a time consuming process. For instance, considering table 5.1, we are going to calculate the game variables for the 183 players that have more shows. Each player has an average of 2605 to process. This means that 476715 hand odds will be calculated. Without using Monte Carlo's Method, each hand odds calculation would take about 5 minutes. This means that to process all the players it would take more than 4 and half years to complete the process. Even using the Monte Carlo's Method, a small reduction in computation time can mean saving a few hours.

The most time consuming part of odds calculation function is the hand evaluation function. For this reason, three evaluators were tested to see which one was the fastest. Table 5.4 shows a summary of the performance results achieve for the evaluators considered.

Table 5.4: Hand evaluator and HoldemML Stats Extractor performance

Hand evaluator	Elapsed time executing hand odds, using Monte Carlo's method with 10.000 trials (ms)	Expected time duration of HoldemML Stats Extractor process (hours)
Alberta Evaluator	800 ms	105,94 hours
Foldem Evaluator	300 ms	39,73 hours
TwoPlusTwo Evaluator	85 ms	11,26 hours

The tests were performed by invoking the function that calculates the odds of an actual hand, with 10.000 Monte Carlo's trials, with three different evaluators. After invoking 1.000 times each function with each evaluator, the average running time of the function was determined. After getting the average running time of hand odds, we can estimate the stats extraction time, based on the data source size. As can be seen, using the TwoPlusTwo hand evaluator it is possible to save lots of hours of processing.

A simple application that reproduces this whole process was created. It gets as input the directory where the HoldemML files are, the player list, and the output dir. After pressing the extract button, the application will determine the game variables for all the players that have at least 500 shows, and save the lists to the prompted output dir.

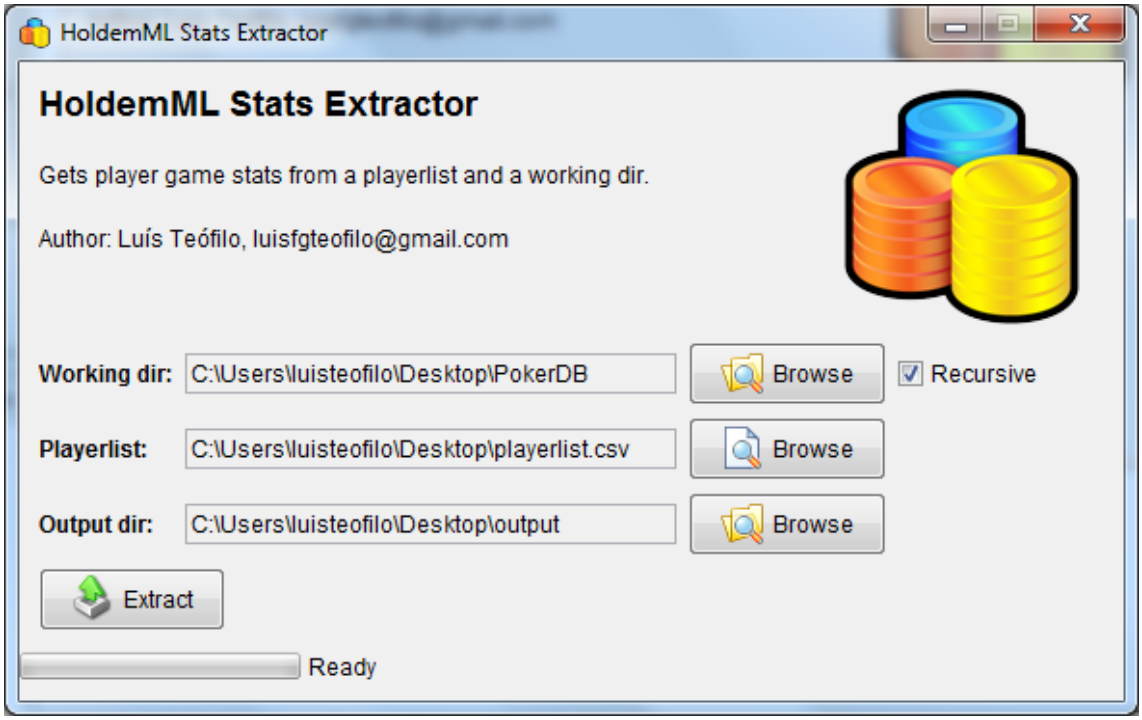

Fig 5.11: HoldemML Stats Extractor

5.7 Summary

In this chapter all the stages of data extraction, processing and preparation were discussed. It was also presented the architecture of the HoldemML modules that were created. There were also demonstrated methods to improve the efficiency of data extraction, namely the importance of having a fast hand evaluator.

6 Learning game Strategies

This chapter presents the application and analysis of the machine learning techniques over the extracted data. To this effect, practical examples of the usage of different data classifiers are demonstrated, over the recovered game variables of some players with different characteristics.

6.1 Testing data

Before turning to the analysis of the classifiers, the test data to be used was defined. The table 6.1 shows eight players with different characteristics that will be used to test the data mining classifiers.

Table 6.1: Characteristics of the players.

Name	*Game Count*	*Money Won*	*Num All-In*	*Num Bets*	*Num Calls*	*Num Flops*	*Num Raises*	*Num Shows*	*Num Wins*
John	15763	1003	65	8025	2133	4881	2837	638	2899
Paul	69750	19087	193	27513	1293	14199	11410	2559	10817
Brian	20739	233	7	6227	3098	4486	1367	761	2068
Jason	97940	6709	204	39376	664	20816	19853	3044	15417
James	43158	16031	128	19343	1580	11212	10196	1401	8420
Kevin	20660	30	87	6384	4799	4714	671	838	2123
David	77598	14142	181	31443	3264	15120	11522	2103	11086
Jeff	33257	-4945	16	21440	3410	11132	7184	882	8159

To be noted that a player with negative earnings was included (Jeff). This will serve for testing purposes later, to check if a tactic generated from a theoretical bad player looses against a tactic from a theoretical good player.

6.2 Player model

Before training the testing model, a player model was defined. The figure 6.1 summarizes the player model.

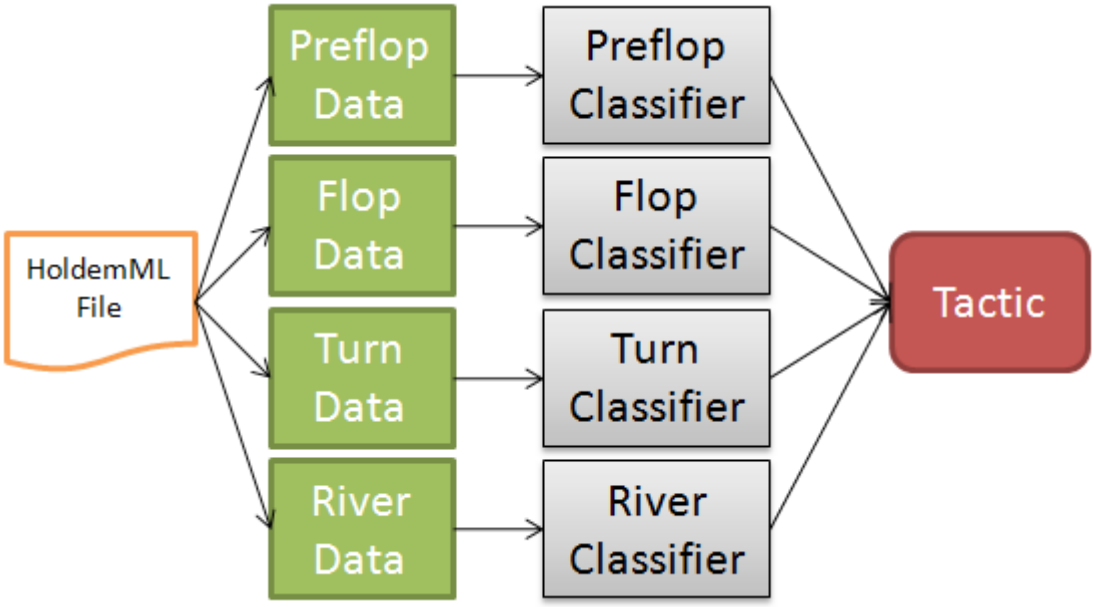

Fig 6.1: Player model

The data was divided into four subsets, each one representing a game round. This way, each tactic is in reality composed by four different tactics. This separation was made because the players might use different tactics in each round. The information on each round is also rather different, because for instance, in the River we probably have much less opponents than in Pre Flop.

6.3 Training Set

The software that was used to train the classifiers was Weka [76]. Before training the classifiers it is necessary to create an ARFF file so Weka can recognize the data. Thus the tables 5.2 and 5.3 generated the following ARFF structure.

```
@relation poker

@attribute positionScore numeric
@attribute isDealer numeric
@attribute handOdds numeric
@attribute handStrength numeric
@attribute percentageMoneyToBet numeric
@attribute percentageMoneyOnTable numeric
@attribute possibleWins numeric
@attribute didLastPlayerBet numeric
```

```
@attribute didLastPlayerRaise numeric
@attribute didLastPlayerAllIn numeric
@attribute didLastPlayerCall numeric
@attribute didLastPlayerCheck numeric
@attribute isBlind numeric
@attribute isLastPlayerAggressive numeric
@attribute isLastPlayerTight numeric
@attribute bet {check, call, allin, raise5, raise10,
raise15, raise20, raise25, raise30, raise50, raise70,
raise90, bet5, bet10, bet15, bet20, bet25, bet30, bet50,
bet70, bet90}
```

Fig 6.2: Training set structure

Three notes on this model:

- In order to facilitate and speed up the learning process, less attributes where chosen than the ones on tables 5.2 and 5.3;
- All input attributes were converted to numeric. Since in the original table we had only real or Boolean attributes, the only ones that had to be converted were Boolean variables. Boolean variables with true value were converted to 1.0 and Boolean variables with false value were converted to 0.0.
- The output attributes were converted to a unique attribute of enumeration type. This will also speed up the learning process as well as reduce the error rate.

6.4 Tactic learning

Having the data ready, the next step is to use classifiers to learn the tactics presented on them. The following classifiers were used:

- Bayesian Networks (BayesNet) - probabilistic graphical model that represents a set of random variables and their conditional dependences via a directed acyclic graph;

- Best First Search Tree (BFTree) – tree search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule;

- C4.5 Search Tree (J48) – updated version of ID3 classifier, that uses the concept of information entropy to create a decision tree;

- Multilayer Perceptron Neural Network (MultilayerPerceptron) – feedforward artificial neural network that maps sets of input data onto a set of appropriate output. This type of network uses the Backpropagation algorithm to train the data;

- Naive Bayes (NaiveBayes) - a simple probabilistic classifier based on applying Bayes theorem with naive independence assumptions;

- Random Forest Search Tree (RandomForest) – classifier that consists of many different decision trees and outputs the class that is mode of the class's output by each individual tree;

- SMO (SMO) – support vector machine with a sequential minimal optimization training algorithm;

- Support Vector Machines (LibSVM) – similar to SMO, but this classifier uses LibSVM library, which means that it runs faster.

The classifiers were evaluated in two steps: learning time and error rate. The following table contains the average error of the four trained classifiers, by name and classifier type. The error was determined using ten-fold cross validation.

Table 6.2: Classifier error rate

	BayesNet	*BFTree*	*J48*	*Multilayer Perceptron*	*NaiveBayes*	*Random Forest*	*SMO*	*LibSVM*
John	0,243	0,195	0,196	0,241	0,332	0,188	0,242	0,251
Paul	0,216	0,193	0,180	0,219	0,283	0,184	0,246	0,268
Brian	0,206	0,128	0,126	0,172	0,242	0,123	0,196	0,224
Jason	0,192	0,152	0,151	0,179	0,24	0,142	0,206	0,227
James	0,321	0,278	0,289	0,351	0,441	0,279	0,409	0,442
Kevin	0,130	0,095	0,097	0,122	0,217	0,095	0,127	0,191
David	0,230	0,191	0,191	0,223	0,312	0,188	0,267	0,307
Jeff	0,233	0,169	0,164	0,185	0,322	0,163	0,214	0,226

The chart on fig 6.3 demonstrates the average error for each classifier.

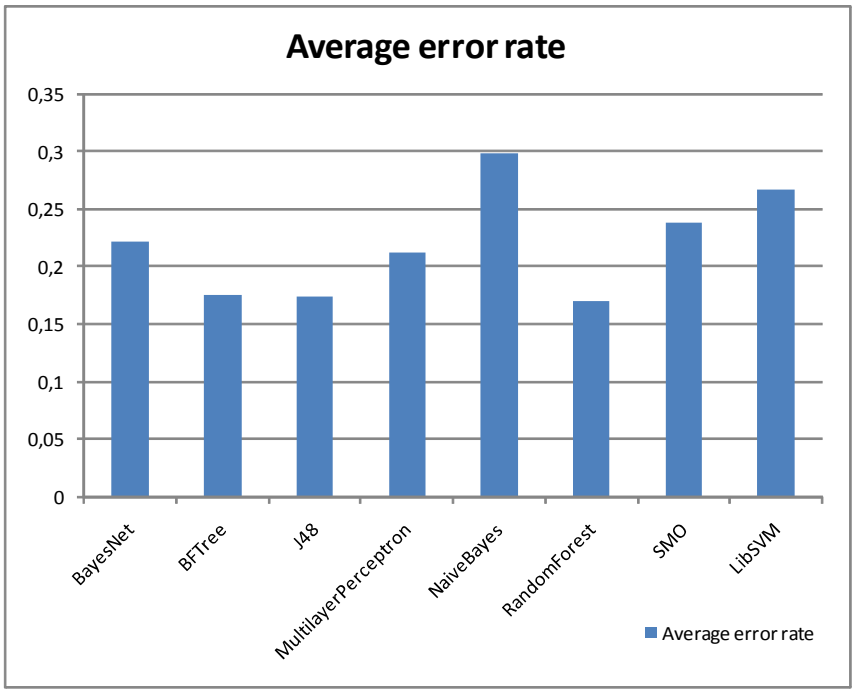

Fig 6.3: Average error rate

As can be seen across the table and graph, the classifiers that performed better for this player model were the search trees, more particularly the Random Forest Trees with an average error of approximately 17%. The errors were relative high as expected as players tend to change tactic during the game, making it difficult to find a pattern with little error.

The classifiers error rate was also analyzed per round. The first player analyzed was the one with less error rates (Kevin). Other players were analyzed with similar results. The results achieved are depicted in figures 6.4, 6.5, 6.6 and 6.7.

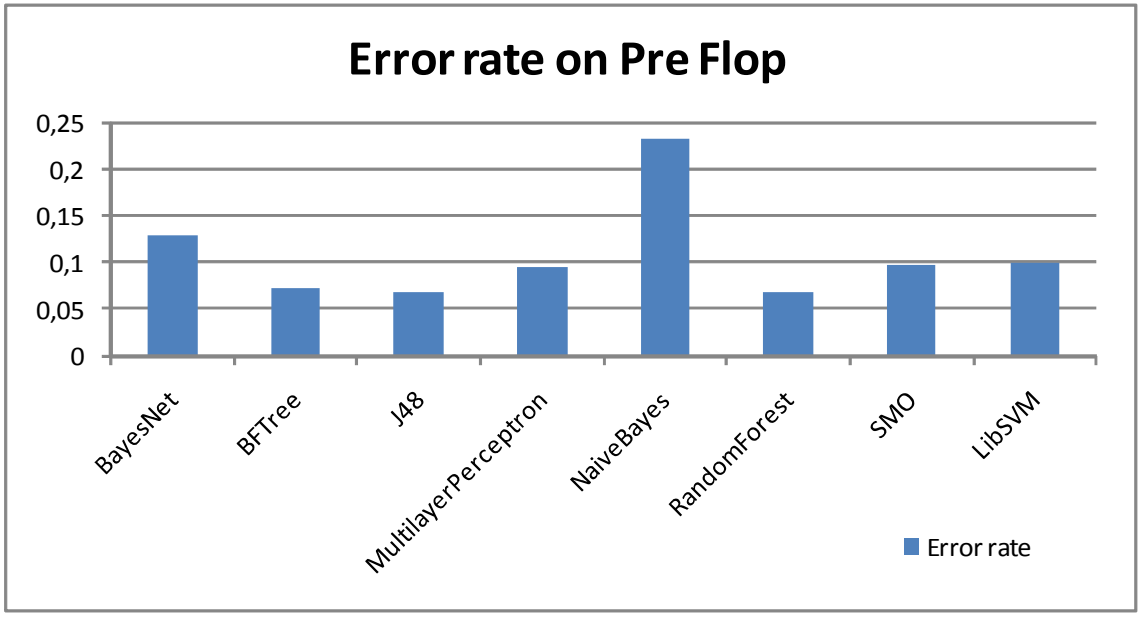

Fig 6.4: Average error on PreFlop (Kevin)

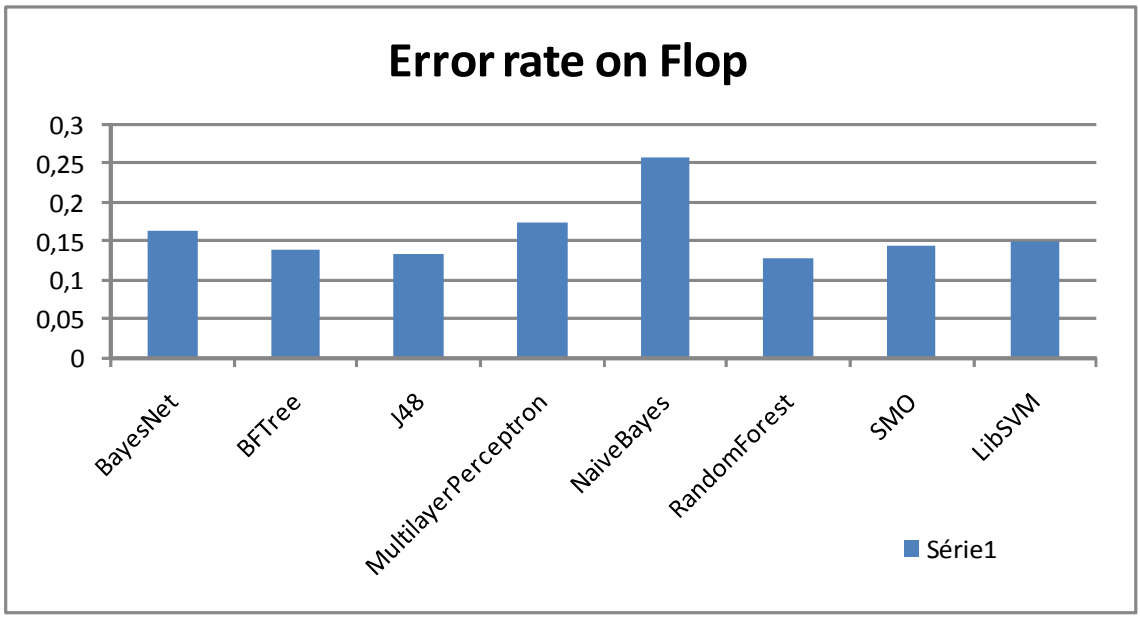

Fig 6.5: Average error rate on Flop (Kevin)

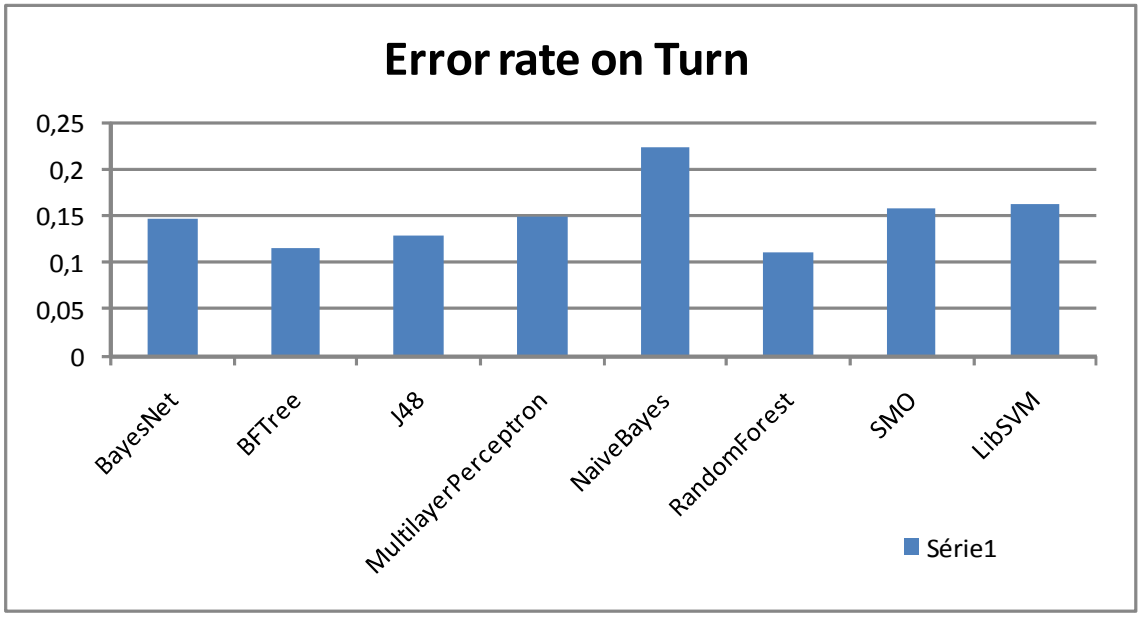

Fig 6.6: Average error rate on Turn (Kevin)

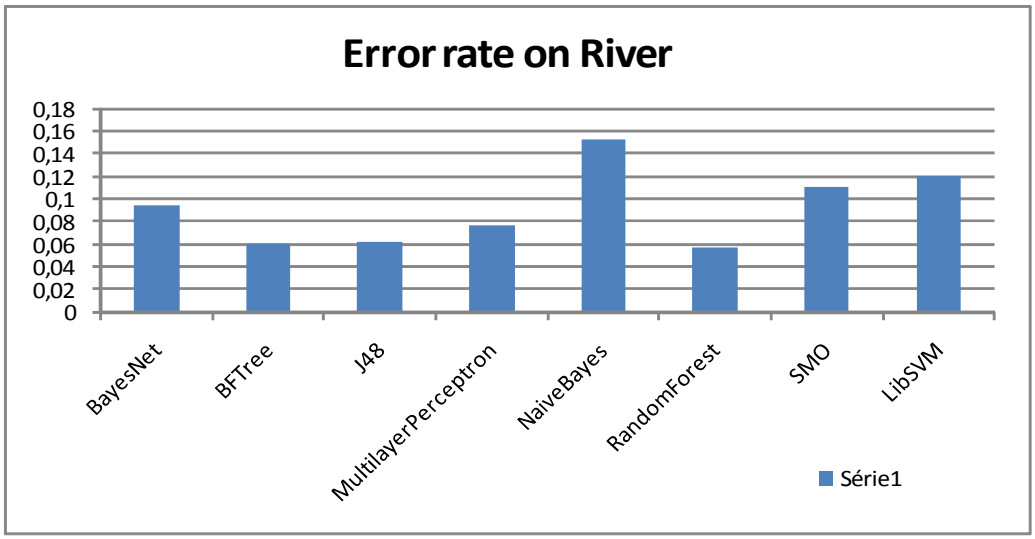

Fig 6.7: Average error rate on River (Kevin)

Again, the error rate of search trees is lower than with other classifiers. For this player the better set of classifiers would be:

- Pre Flop: J48 / Random Forest
- Flop: Random Forest
- Turn: Random Forest
- River: Random Forest

Another fact that can be observed is that the error is much higher in Flop and Turn round than in Pre Flop and River rounds. This was expected for Pre Flop round, since there are no community cards the tactics tend to be simpler. As for River round, this can be explained by the lower number of players on that round. The following chart summarizes the error evolution on game rounds.

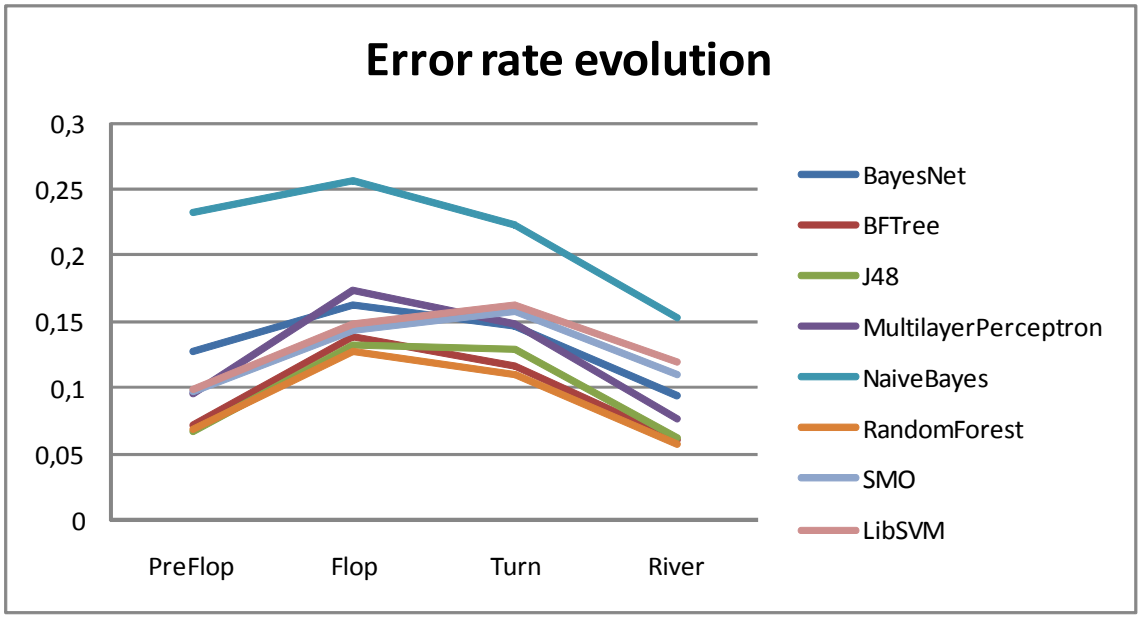

Fig 6.8: Average error rate on River

Regarding the training time, the following measures were taken.

Table 6.3: Classifier training time (ms)

	BayesNet	*BFTree*	*J48*	*Multilayer Perceptron*	*NaiveBayes*	*Random Forest*	*SMO*	*LibSVM*
John	117	1129	191	110901	43	692	10251	552
Paul	187	6864	2162	214270	654	7030	19345	8881
Brian	136	1366	396	274955	398	2299	17018	6120
Jason	281	10792	580	362997	1324	7287	22732	6632
James	346	1820	482	80696	377	3197	13931	4672
Kevin	162	1080	220	161503	254	633	13134	2515
David	761	3190	825	248757	508	5950	18111	9465
Jeff	143	1721	1360	158291	233	807	12666	3710

The following chart demonstrates the average training time for each classifier. The Multilayer Perceptron classifier was removed from the chart because its training time is much higher than the others.

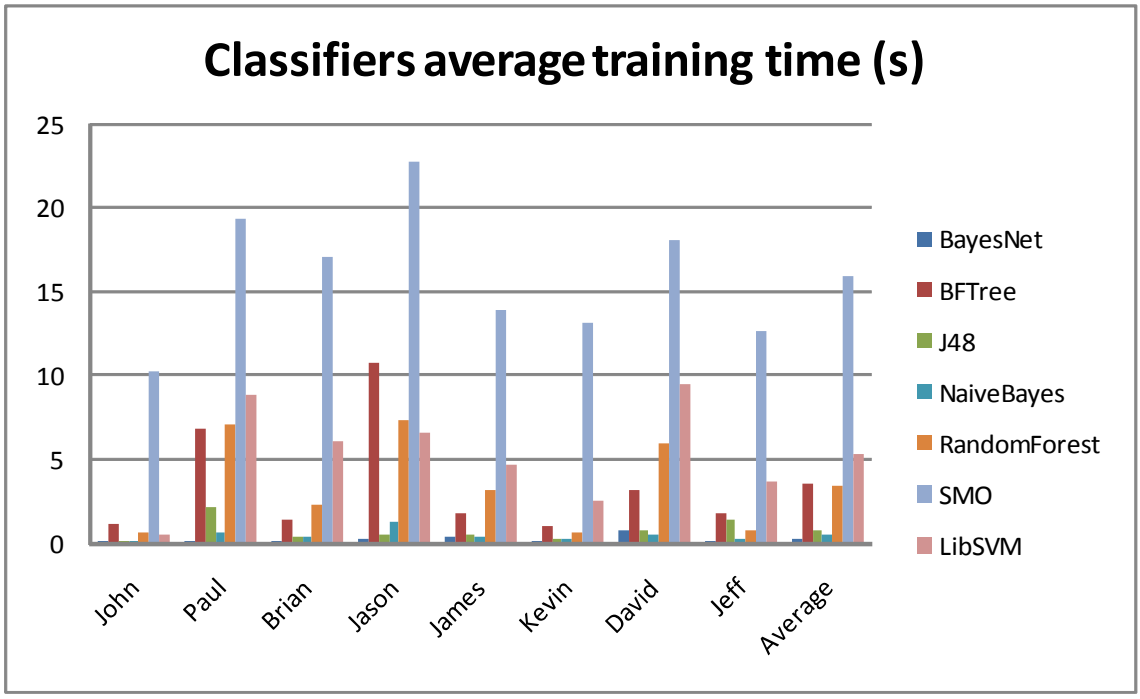

Fig 6.9: Average training time (s)

As it can be seen, the classifiers based on Bayesian inference are more efficient in the training phase. Neural networks and SVM classifiers are the slowest ones. We can conclude that the time saved in training the Bayesian classifier is not significantly large to be passed over the search trees that have significantly lower error rates.

6.5 Strategies

After learning the tactics, the next stage is to implement the agent strategy. Since the tactics lacks complex opponent modelling, if the agent were to use the tactics alone, it would be crushed by any opponent that does opponent modelling. So, the idea behind the strategy is to confuse the opponent's by using several different tactics.

The agent strategy can be represented by the following figure.

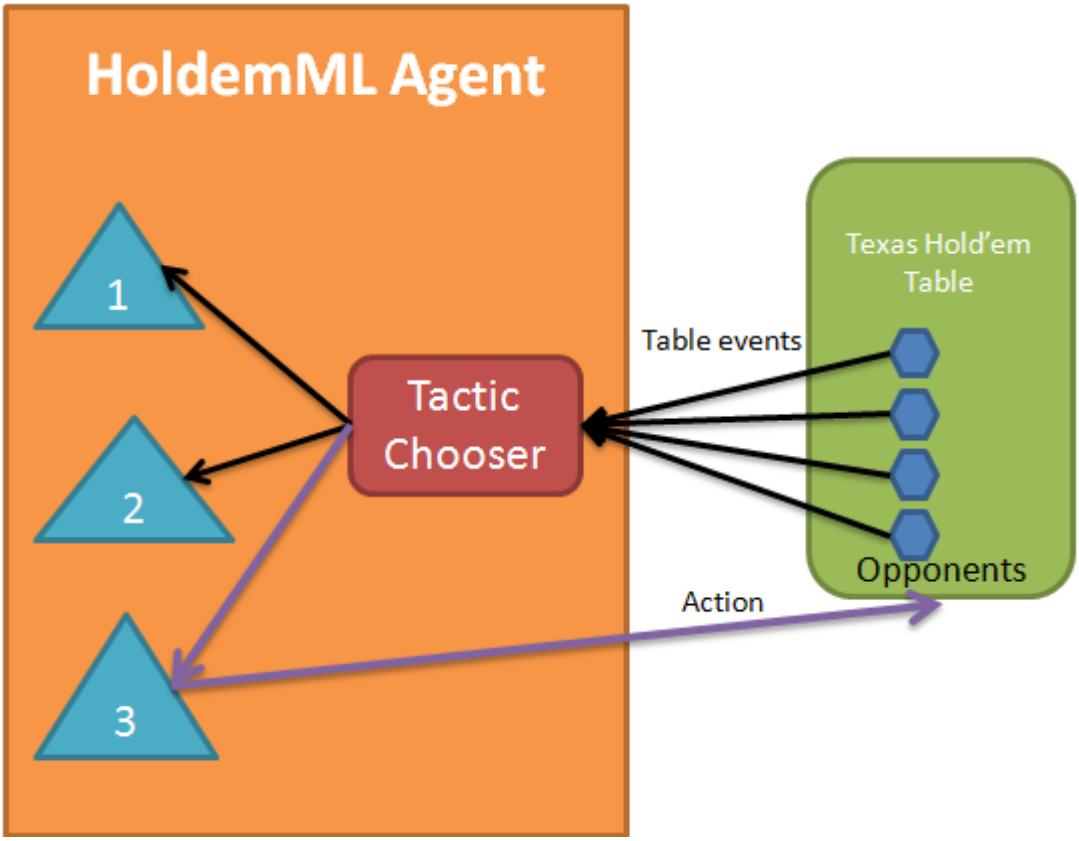

Fig 6.10: HoldemML agent strategy

As it can be seen, the agent based on the table events, chooses its tactic. Then the tactic, based on the game variables, provides a prediction of the action and the agent executes that action.

Four simple tactics were defined:

- **No change strategy**: the agent chooses a random tactic at the start of the game and never changes it.
- **Periodic change strategy**: the agent changes its tactic from 10 to 10 plays.
- **Random change strategy**: after every action, the agent changes its tactic.
- **Simple change strategy**: the agent chooses a random tactic at the start of the game. If the earnings are low with the current tactic, the agent changes to another random tactic.

6.6 HoldemML Strategy generator

After learning the tactics and defining the strategies, we can now create and agent. The API used to implement the agent was the Meerkat API. The agent is then composed by two files:

- **Meerkat agent configuration file**: every Meerkat agent must have a configuration file. This configuration file tells which Java class implements the agents, and in this particular case it indicates the path where the strategy file is stored.
- **Strategy file**: a file that contains the trained classifiers.

An application to generate these two files was created.

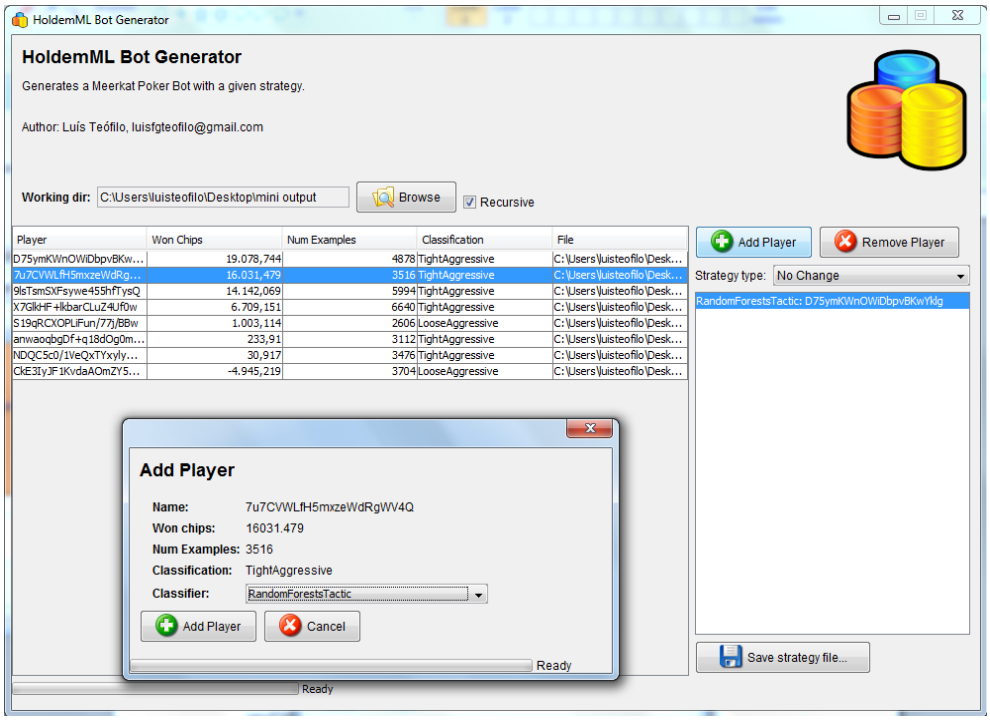

Fig 6.11: HoldemML Bot Generator

The first step to use this application is to load a directory with stat files built by HoldemML Stats Generator. After that, we can choose which players tactics will be used on the strategy. We add the players using add button, and after that choosing which classifier will be used to train the player tactic. After choosing all the players that compose the strategy, we can choose the strategy type, which defines the heuristic of tactic change.

6.7 HoldemML Meerkat Agent

The implemented agent uses a strategy file (indicated by the agent configuration file) to guide its actions. The way an action is calculated can be given by the following pseudo-code.

```
Action getAction(chipsToCall, remainChips, table, strategy)
{
   //!First step
   canCheck = (chipsToCall == 0);
   canRaise = (chipsToCall > 0 && remainChips > chipsToCall);
   canBet = (chipsToCall == 0 && remainChips > table.bigblind);
   canCall = (chipsToCall > 0 && remainChips >= chipsToCall);
   canAllIn = true;

   //!Second step
   strategy.tryChangeTactic(table);

   //!Third step
   action = predictAction(strategy.currentTactic.classifiers,
                          table, canCheck, canRaise,
                          canBet, canCall, canAllIn,
                          strategy.currentTactic.tightFactor);

   //!Fourth step
   if(action == null) { /* if the classifier can't predict the
action */
      return new CheckOrFold();
   }

   return action;
}

Action predictAction(classifiers, table, canCheck, canRaise,
canBet, canCall, canAllIn, tightFactor)
{
   //!Fold condition
   if(table.handStrength < 0.5 && Random(0,100) > tightFactor) {
      return new CheckOrFold();
   }

   result = classifiers[table.currentRound].getClassPredictions();
   max = 0.0;
   classIndex = 0;

   for(i = 0; i != result.length; ++i) {
      if(result[i].type is Check && canCheck || result[i].type is
         Raise && canRaise || result[i].type is Bet && canBet ||
         result[i].type is Call && canCall || result[i].type
         is AllIn && canAllIn) {

           if(result[i].value > max) {
```

```
                classIndex = i;
                max = result[i].value;
            }
        }
        if(max < min_acceptance_val) {
            return null;
        }
        return result[classIndex].action;
}
```

Fig 6.12: HoldemML Agent Action Choosing Algorithm

The way the agent chooses the action based on the current game state can be divided into four steps:

- First, the agent determine which actions can be applied;
- Try to change tactic based on the game state. The strategy class is in charge of this operation.
- Uses the classifiers to predict the action.
- If the prediction is null i.e. the classifier can't find any action that suits the situation, the agent try to check. If it is not possible to check, than the agent folds the hand.

Regarding the action prediction, since the classifiers don't support recognition of hands to Fold, because the hands used to teach the agent are show hands, it is necessary to define criteria to fold some hands. The defined criteria was that if the hand strength is below 50%, than the agent has a probability of folding equal to its tightness level.

A classifier returns a double between 0 and 1 for each bet class, which means the acceptance level of that class. Therefore, to extract the correct action we need to see which action is possible and has the higher acceptance level. Before returning, the max acceptance value is checked to see if it fulfils a predefined minimum value.

6.8 HoldemML Simulator

To test all the agents, a simulation application was created. This application is mostly a wrapper of Meerkat Open Test Bed, to facilitate the setting up of the table.

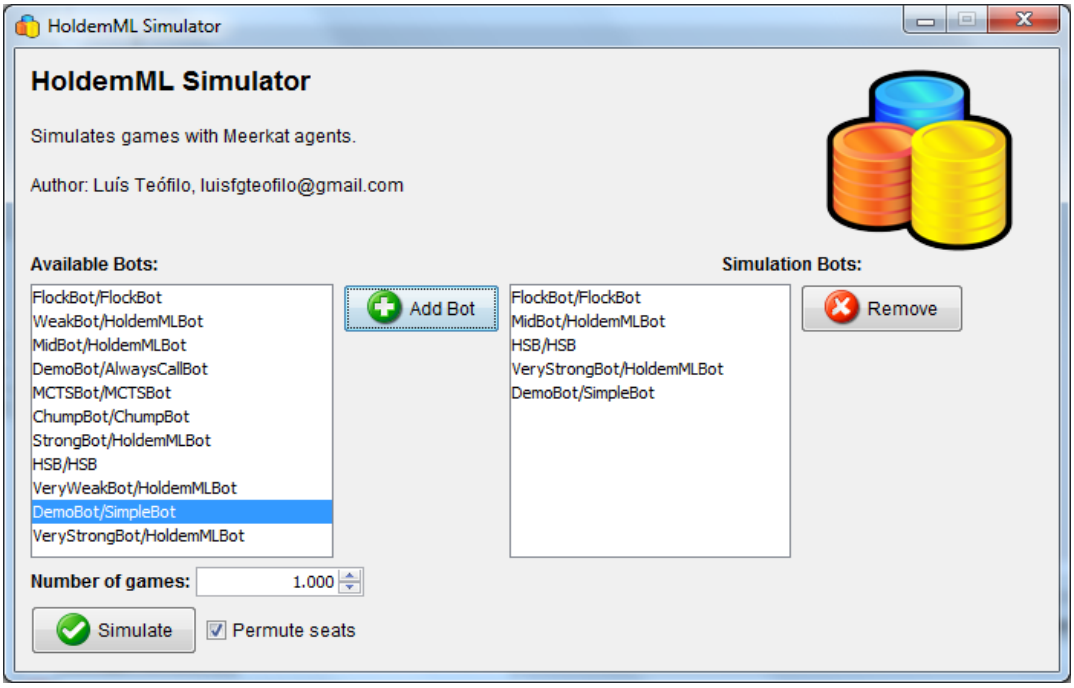

Fig 6.13: HoldemML Simulator

The produced application is very easy to use. First, we have a list of available bots. From that list we can choose which bots will participate in the simulation. After choosing the participating bots, we choose the number of games and if we want to activate seat permutation. Seat permutation allows reducing variance in testing, by saving repeating the same experience with the agents in different seats. Finally, we hit the simulate button and wait for the simulation to finish. The application shows a chart that shows the money of each agent at each instant, while simulating the games.

6.9 Summary

In this chapter all the steps needed for the agent creation were described since tactic learning, classifier evaluation, strategy usage and determine the next agent's move. All the applications built for that purpose were also presented and discussed.

7 Experiments and results

This chapter presents the experiments performed using the agents developed and the results achieved on those experiments, as well as their discussion.

7.1 Poker Agent testing

After creating the agent and defining some strategies, the agents were tested to validate this poker agent building approach. The tests were based on strategies created from tactics provided by the players on table 6.1.

All tests were made with the built application HoldemML Simulator, with cash games, with 1.000 games per test and table seat permutation.

Next, the description and test results will be presented.

7.1.1 Inner testing

This section presents tests between HoldemML agents.

The first test opposed David against Jeff, respectively a player that has big winnings against a player who has big losses. The simulation results were as follows.

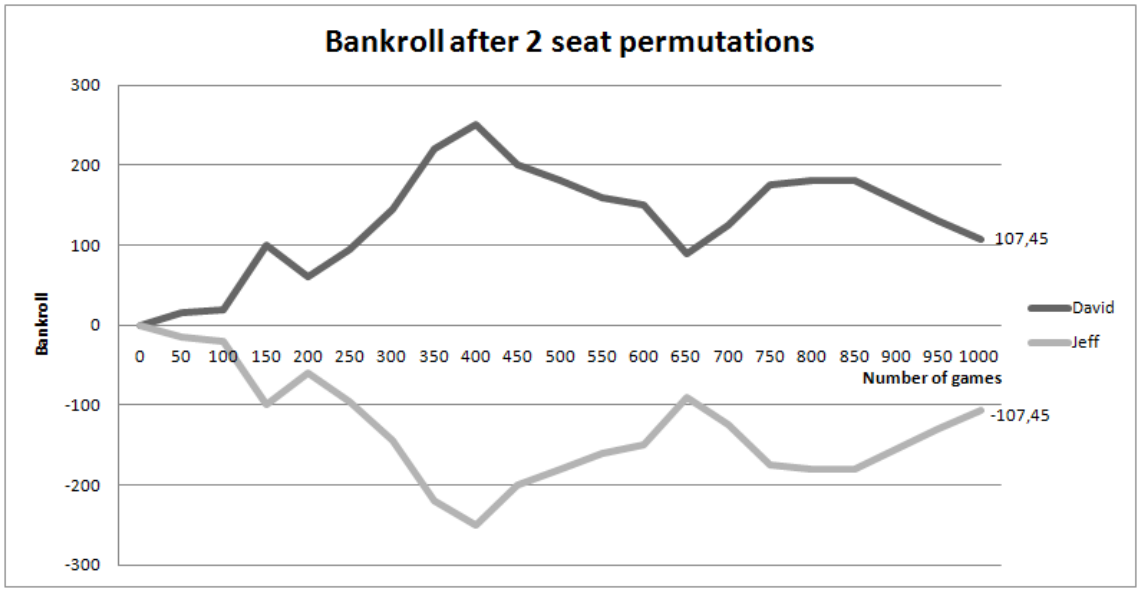

Fig 7.1: HoldemML David VS HoldemML Jeff

As was more probable, the player with more winnings (David) won by a large margin (214,90$) against Jeff.

Another test was done opposing the same player Jeff against Kevin, a player who has broke even. The results were as follows.

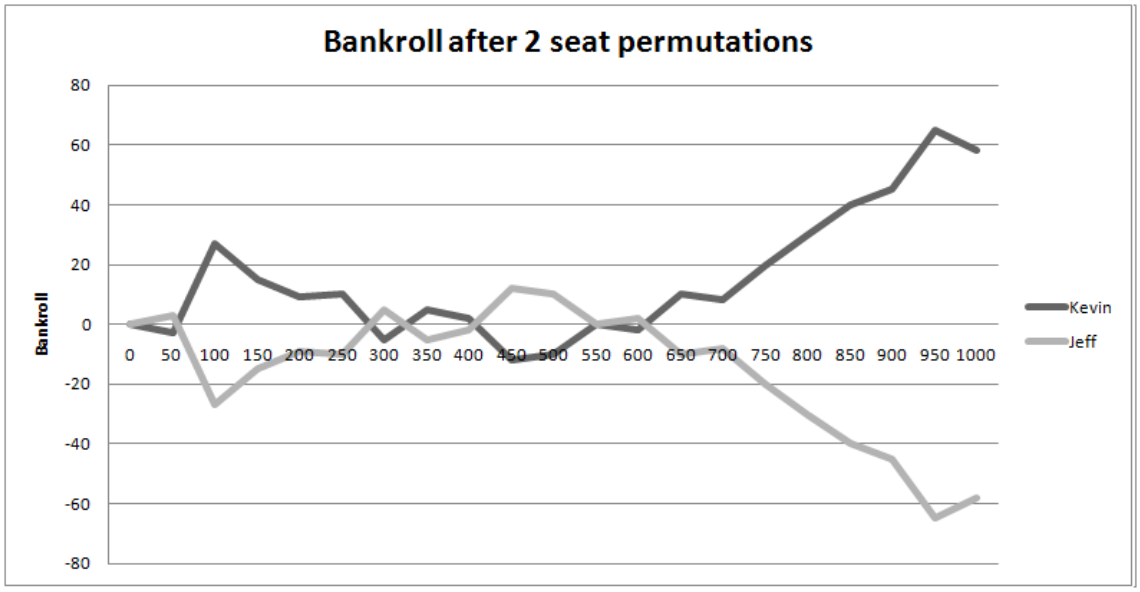

Fig 7.2: HoldemML Kevin VS HoldemML Jeff

Again, for the same reason as the last experiment, Jeff lost again but now for a lower margin (116,30$), which was expected because Kevin is better player than Jeff in real world but is not as good as David.

Finally, a test with four agents was made to evaluate the behaviour of the agents against multiple players.

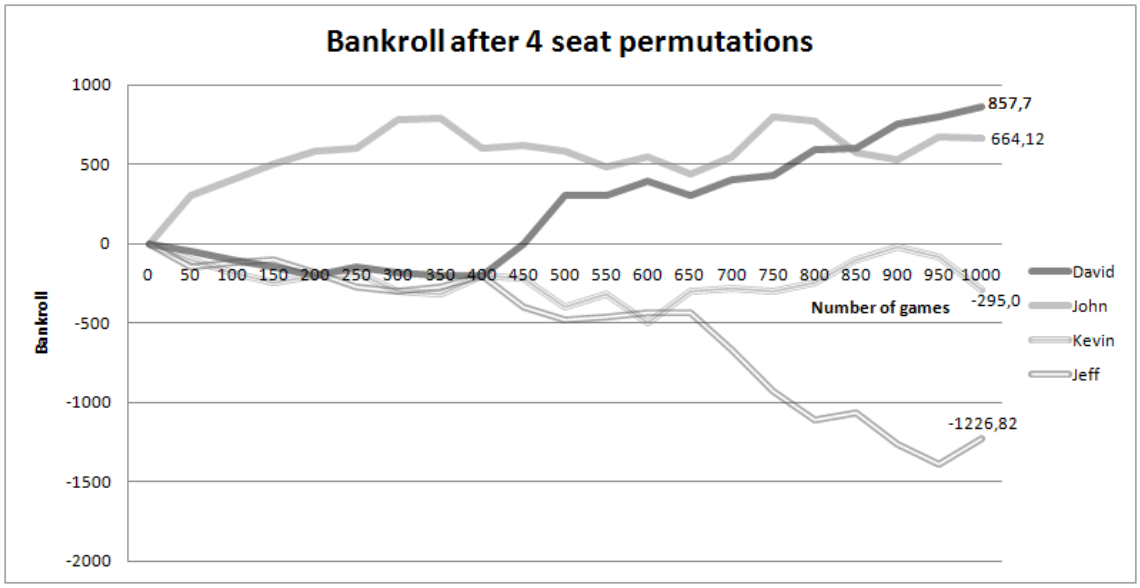

Fig 7.3: Multiple HoldemML agents test

As it can be observed, the conclusion is the same as before. The player David, is the one who won more money in this simulation, having a total profit of 857,70$. Then, the agent John with a profit of 664,12$. After John, Kevin with looses of -295,00$ and Jeff, which was the worst player, with looses of -1.226,92$.

7.1.2 Outer testing

Tests against other bots were also done. The first test was trivial and opposed Paul against an Always Call bot.

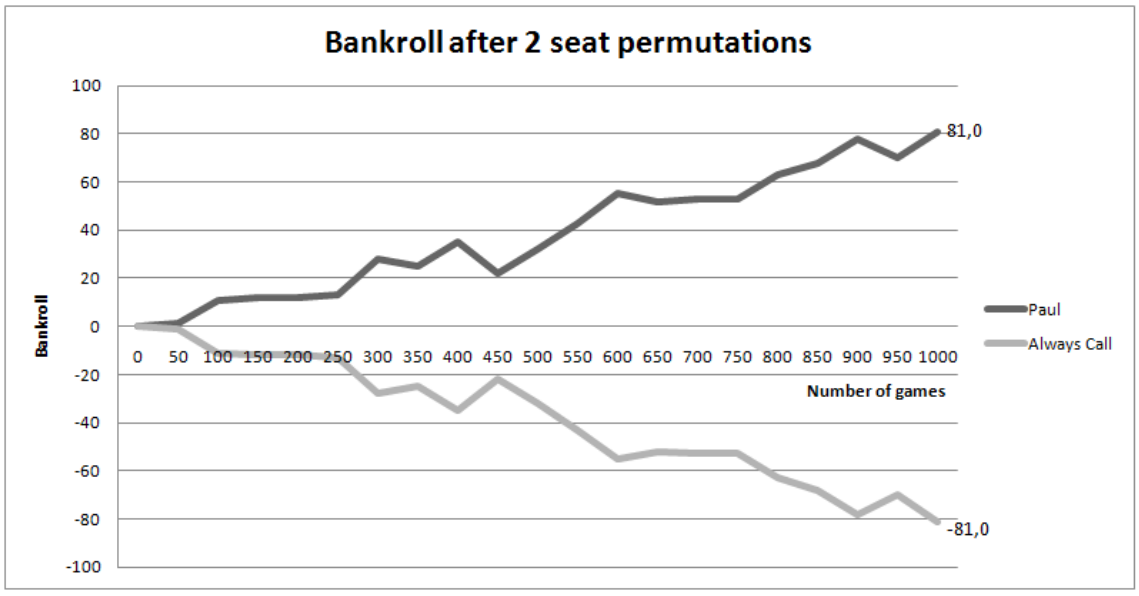

Fig 7.4: HoldemML Paul VS Always Call Bot

As it was expected, the Always Call bot was beaten by a good margin (162,00$) since it never folds a hand.

The next test opposed Paul against Hand Strength Bot. The strategy of hand strength bot is only based on the current hand strength of the agent. The rules of its strategy are as follows:

- If the hand strength value is below 30%:
  - The agent calls or checks (30% probability);
  - The agent checks or folds (70% probability).
- If the hand strength value is higher or equal than 30% and bellow 50%:
  - The agent raises 25% of the pot (100% probability).
- If the hand strength value is higher than 50%:
  - The agent raises half of the pot (50% probability);
  - The agent goes all-in (50% probability).

The results of the match between Paul and Hand Strength Bot are as follows.

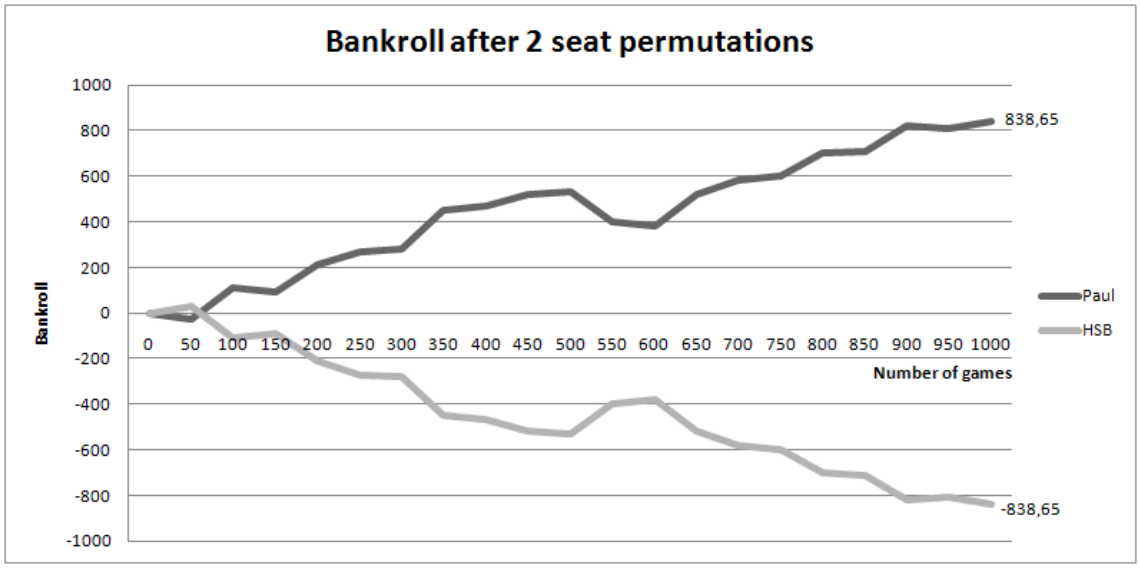

Fig 7.5: HoldemML Paul VS Always Call Bot

As it can be observed, Paul has won again by a very good margin (1677,30$). We can conclude that a HoldemML agent based on a good real player can beat agents with simple strategies.

After testing the agent against other weak agents, the agent was tested against an agent that was capable of opponent modelling : MCTS Bot. The MCTS Bot builds a Monte-Carlo-Tree-Search to decide upon its actions. It models the opponent using search trees classifiers. To determine the next action, first it checks the opponent tree and predicts which actions the opponent might take. The MCTS Bot chooses its action based on those predictions.

The results of the match between Paul and MCTS Bot were as follows.

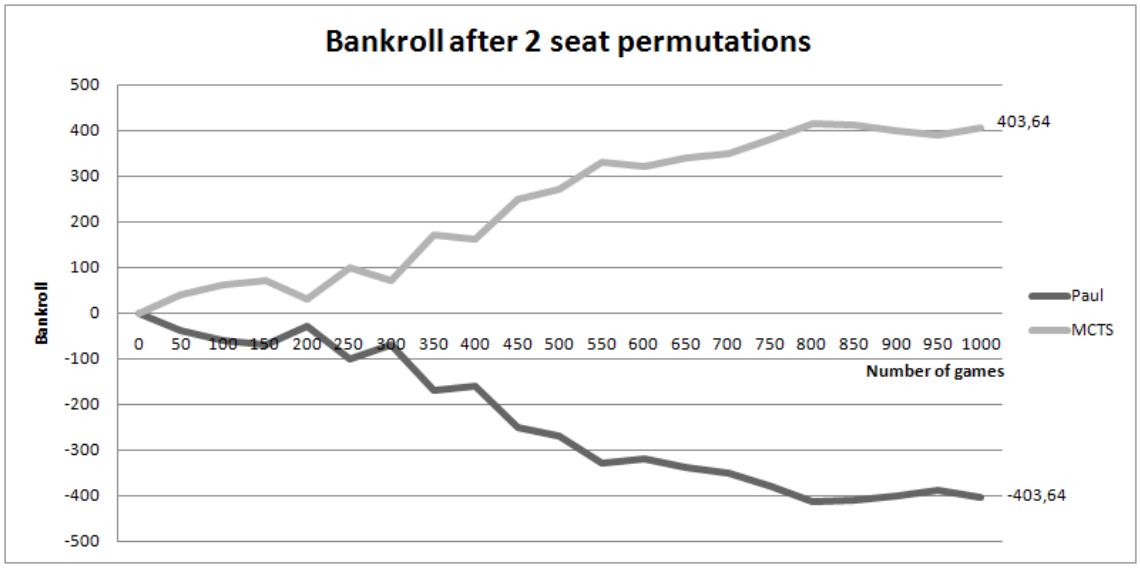

Fig 7.6: HoldemML Paul VS MCTS Bot

As it can be observed, Paul has lost by a very large margin (807,28$). This happened because a static strategy can't compete against a strategy with opponent modelling. After the MCTS Bot modelled Paul's strategy (near the 300th game), the MCTS Bot winnings start to increase.

7.1.3 Behaviour testing

During all these tests, the actions of the agents were recorded. It was verified that, regarding the Slansky classification, the agents had similar classification to the real players that led to their tactics. This may be verified on table 7.1.

Table 7.1: Agent behaviour VS real player behaviour

Player	*Agent aggression factor*	*Real player aggression factor*	*Agent tightness*	*Real player tightness*	*Agent classification*	*Real player classification*
John	4,77	5,12	0,32	0,31	LooseAgressive	LooseAsgressive
Paul	22,34	30,25	0,22	0,20	TightAgressive	TightAgressive
Brian	5,44	2,45	0,25	0,22	TightAgressive	TightAgressive
Jason	70,22	89,51	0,19	0,21	TightAgressive	TightAgressive
James	18,89	18,78	0,28	0,26	TightAgressive	TightAgressive
Kevin	1,55	1,49	0,25	0,23	TightAgressive	TightAgressive
David	16,04	13,22	0,19	0,19	TightAgressive	TightAgressive
Jeff	9,42	8,40	0,44	0,33	LooseAgressive	LooseAgressive

The resulting errors were expected. First, the simulated games are different from the games on the data source. Moreover, the agent's tactic was not fully modelled because it wasn't possible to correctly model when the agent should fold the hand, due the lack of that information on the data source.

Regarding aggressiveness and tightness of the players, it was found that the players with higher aggression error are Paul and Jason and the player with higher tightness error is Jeff. There isn't enough information to get any conclusions of that fact. The reason is possibly related to the complexity of the tactics used by the players, or eventually the players changed tactic during the games.

7.2 Strategy testing

After testing tactics individually, the next step is to test strategies that combine tactics to see if the results improved.

The defined strategy combined the best extracted tactics: which are the ones from the players David, Paul and James, which are the players that won more money. The strategy used was SimpleStrategy, which is a strategy that changes tactic when the agent is losing money. The agent, named MegaBot, was tested against MCTS Bot and the results were as follows.

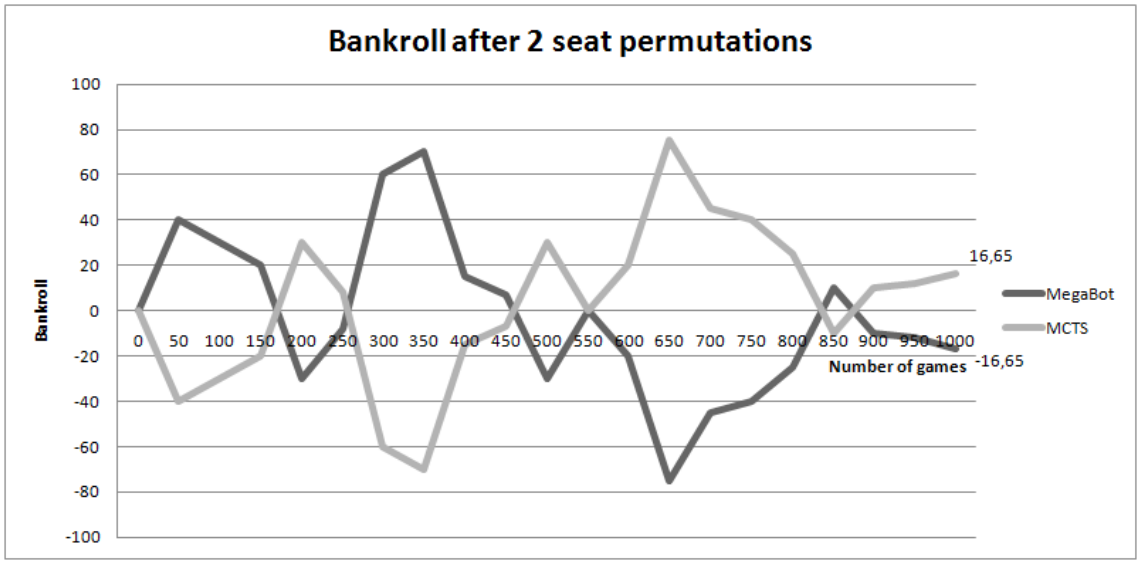

Fig 7.7: HoldemML MegaBot VS MCTS Bot

As it can be observed, even though the MegaBot have lost against MCTS Bot, the results improved a lot against using a one tactic strategy. MegaBot has lost by a very low margin.

Note the occurrence of cycles in the chart. First, the MegaBot begin to earn money until the MCTS learn the current MegaBot tactic (iteration 150). When MegaBot starts losing money (iteration 200), it makes a tactic change, which confuses the agent MCTS, giving advantage to MegaBot (iteration 250).

7.3 Summary

In this chapter it was demonstrated the behaviour of HoldemML agent against other agents, using different tactics and strategies. It was verified that tactics from agents that won more money in the past were best than tactics that won less money, or even loose money. It was also verified that the use of different tactics through the game improves the agent results against opponents that are capable of opponent modelling.

8 Conclusions

This chapter presents the conclusions of this research, through confrontation against the defined goals. It also presents future work about this domain.

8.1 Goal achievement

The main purpose of this research work was to create a poker agent based on the observation of past human games, to see if it was possible to create a competitive poker player by that means.

First, a common format for representation of poker hand history was created and named HoldemML. It is a simple XML type format that might have potential to replace the actual formats, since as well as being readable for humans; it has a structure associated with it, making it easier for machines to process its information. Furthermore, there are already plenty of tools to process XML documents.

After defining the common format, a significant amount of poker hand data was extracted and converted to that format. From the recovered data, it was extracted game state information about the players' actions. Using that information, game state variables that influence the players action, were defined. While extracting the game state information it was noted that the use of an efficient hand evaluator is essential to optimize the process. Some hand evaluators were analyzed and it was concluded that TwoPlusTwo Evaluator was the most efficient.

After extracting the game variables, various data mining classifiers were used to learn the player's tactic, to see which one was the most efficient both in error minimization and learning time. It was concluded that, for the defined player model, the search trees were the best classifiers, specially the Random Forest Tree which had the smaller error rate. The quicker

classifiers were the one's based on Bayesian inference, but the time difference for search trees does not justify the higher error rate.

The generated tactics were proven to represent quite well the real players. After testing the tactics it was verified that the best real player tactics have beaten the worst player tactics. It was also concluded, from the experiments conducted, that the aggressive factor and tightness of the agents remained very similar to the same variable of the real modelled players.

It was also that the aggressive factor and tightness of the players remained similar.

After training the tactics, simple strategies were defined to compensate the lack of opponent modelling on this agent. After testing the agent, it was verified that changing the tactic during the game, against an opponent that does opponent modelling, can improve the results. However, an agent without opponent modelling can't be competitive against good poker players, as the agent in spite of improving its results, has still lost the match.

To conclude, a fully working framework was created, that enables anyone to easily create Poker Agents just by providing game logs.

8.2 Future work

There is still a long way to go to create an agent that plays poker at the best human players. This research presented an approach based on supervised learning methodologies, where the agent copies past human experience to decide its actions. The created agent is not competitive against very good poker players, but using the created framework with some modifications, the agent model can be greatly improved. Different game variables can be defined, as well as, more complex strategies and integration of opponent modelling might improve the results achieved.

Bibliography

[1] McCarthy, John, “What is artificial intelligence?” John McCarthy's Home Page http://www-formal.stanford.edu/jmc/whatisai/whatisai.html (cited: January 28, 2010).

[2] Gordon E. Moore, Cramming more Components onto Integrated Circuits, Electronics, 38(8), April 9, 1965.

[3] Kurzweil, Ray, The Singularity Is Near: When Humans Transcend Biology. (Penguin Books, 2006).

[4] H. Markram, “The blue brain project,” Proceedings of the 2006 ACM/IEEE conference on Supercomputing, Tampa, Florida: ACM, 2006, p. 53.

[5] NEWSWEEK.com, “Going all in for online Poker.” http://www.newsweek.com/id/56438 (cited: January 18, 2010).

[6] “Poker Channel Europe,” http://www.pokerchanneleurope.com/ (cited: February 12, 2010).

[7] Billigs, Darse. Ph.D. dissertation. Algorithms and Assessment in Computer Poker. Department of Computing Science, University of Alberta, Canada, 2006.

[8] J. Doughney e T. Kelleher, “The impact of poker machine gambling on low-income municipalities,” A Critical Survey of Key Issues. Victoria University of Technology, 1999.

[9] McKenna, James A., *Beyond Tells: Power Poker Psychology*, illustrated edition (Stuart (Lyle) Inc.,U.S., 2005).

[10] G. Smith, M. Levere, e R. Kurtzman, “Poker Player Behavior After Big Wins and Big Losses,” Management Science., vol. 55, 2009, pp. 1547-1555.

[11] D. Sklansky, The Theory of Poker: A Professional Poker Player Teaches You How to Think Like One, Two Plus Two, 2002.

[12] Sêca, Rui André, An Intelligent Poker-Agent for Texas Hold’em. Faculdade de Engenharia da Universidade do Porto. Porto, 2008. Master thesis.

[13] “Computer Poker Research Group Homepage”, http://webdocs.cs.ualberta.ca/~games/poker/ (cited: January 14, 2010).

[14] Kan, Morgan. M.Sc. thesis. Postgame Analysis of Poker Decisions. Department of Computing Science, University of Alberta, Canada, 2007.

[15] A. Davidson. Opponent modelling in poker. Master´s thesis, Department of Computing Science, University of Alberta, 2002.

[16] A. Davidson, D. Billings, J. Schaeffer, and D. Szafron. Improved opponent modelling in poker. In International Conference on Artificial Intelligence, ICAI´00, pages 1467–1473, 2000.

[17] Billings, D., Burch, N., Davidson, A., Holte, R., Schaeffer, J., Schauenberg, T., and Szafron, D. 2003. Approximating game-theoretic optimal strategies for full-scale poker. In *Proceedings of the 18th international Joint Conference on Artificial intelligence* (Acapulco, Mexico, August 09 - 15, 2003). International Joint Conference On Artificial Intelligence. Morgan Kaufmann Publishers, San Francisco, CA, 661-668.

[18] A. Davidson, Using Artificial Neural Networks to Model Opponents in Texas Hold'em, Alberta University, 1999

[19] Darse Billings , Aaron Davidson , Jonathan Schaeffer , Duane Szafron, The challenge of poker, Artificial Intelligence, v.134 n.1-2, p.201-240, January 2002

[20] Gilpin, A. and Sandholm, T. 2007. Better automated abstraction techniques for imperfect information games, with application to Texas Hold'em poker. In *Proceedings of the 6th international Joint Conference on Autonomous Agents and Multiagent Systems* (Honolulu, Hawaii, May 14 - 18, 2007). AAMAS '07. ACM, New York, NY, 1-8. DOI= http://doi.acm.org/10.1145/1329125.1329358

[21] Gilpin, A. and Sandholm, T. 2006. A Texas Hold'em poker player based on automated abstraction and real-time equilibrium computation. In *Proceedings of the Fifth international Joint Conference on Autonomous Agents and Multiagent Systems* (Hakodate, Japan, May 08 - 12, 2006). AAMAS '06. ACM, New York, NY, 1453-1454. DOI= http://doi.acm.org/10.1145/1160633.1160911

[22] Gilpin, A. and Sandholm, T. 2006. A competitive Texas Hold'em poker player via automated abstraction and real-time equilibrium computation. In *Proceedings of the 21st National Conference on Artificial intelligence - Volume 2* (Boston, Massachusetts, July 16 - 20, 2006). A. Cohn, Ed. Aaai Conference On Artificial Intelligence. AAAI Press, 1007-1013.

[23] Miltersen, P. B. and Sørensen, T. B. 2007. A near-optimal strategy for a heads-up no-limit Texas Hold'em poker tournament. In *Proceedings of the 6th international Joint Conference on Autonomous Agents and Multiagent Systems* (Honolulu, Hawaii, May 14 - 18, 2007). AAMAS '07. ACM, New York, NY, 1-8. DOI= http://doi.acm.org/10.1145/1329125.1329357

[24] Johanson, M. and Bowling, M. 2009. Data Biased Robust Counter Strategies. Proceedings of the Twelfth International Conference on Artificial Intel-ligence and Statistics (AISTATS-09).

[25] Kevin Waugh, Nolan Bard, and Michael Bowling. Strategy Grafting in Extensive Games. In Advances in Neural Information Processing Systems 22, 2010.

[26] Beattie, B., Nicolai, G., Gerhard, D., and Hilderman, R. J. 2007. Pattern Classification in No-Limit Poker: A Head-Start Evolutionary Approach. In Proceedings of the 20th Conference of the Canadian Society For Computational Studies of intelligence on Advances in Artificial intelligence (Montreal, Quebec, Canada, May 28 - 30, 2007). Z. Kobti and D. Wu, Eds. Lecture Notes In Artificial Intelligence, vol. 4509. Springer-Verlag, Berlin, Heidelberg, 204-215. DOI= http://dx.doi.org/10.1007/978-3-540-72665-4_18

[27] Google Finance, “bwin Interactive Entertainment AG.”, http://www.google.com/finance?q=WBAG:BWIN (cited: January 14, 2010).

[28] Ferreira, João Carlos Leite, Opponent Modeling in Texas Hold’em: Learning Pre-Flop strategies in multiplayer tables. Faculdade de Engenharia da Universidade do Porto. Porto, 2008. Master Thesis.

[29] Michael Maurer's, “IRC Poker Database.”, http://games.cs.ualberta.ca/poker/IRC/ (cited: January 14, 2010).

[30] “PokerTracker - Online Poker Tracking & Analysis Software Tool”, http://www.pokertracker.com/ (cited: January 14, 2010).

[31] “The free poker database - Homepage”, http://fpdb.sourceforge.net/ (cited: January 16, 2010).

[32] “PokerStars”, http://www.pokerstars.com/ (cited: January 16, 2010).

[33] “Full Tilt Poker”, http://www.fulltiltpoker.com/pt/ (cited: January 16, 2010).

[34] “PokerFTP Hand History Database”, http://pokerftp.com/ (cited: January 14, 2010).

[35] W3C, “Extensible Markup Language (XML)”, Http://www.w3.org/XML (cited: January 25, 2010).

[36] “XML Tutorial”, W3Schools.com, Http://www.w3schools.com/xml/ (cited:January 25, 2010).

[37] Shanky Technologies, “Poker Programming Language User Guide”, 2009. Available online at http://www.bonusbots.com/PPLguide.pdf

[38] Mendes, Pedro Daniel da Cunha, High-Level language to build Poker Agents. Faculdade de Engenharia da Universidade do Porto. Porto, 2008. Master thesis.

[39] sadg, “A Nash Equilibrium in No-Limit Texas Hold’em.” Expert Voices Gateway http://expertvoices.nsdl.org/cornell-info204/2008/02/29/a-nash-equilibrium-in-no-limit-texas-hold%E2%80%99em/ (cited: January 15, 2010).

[40] Jones, Lee H., “Are You Sage? Getting an Edge in Heads-Up No-Limit Hold'em.” CardPlayer.com http://www.cardplayer.com/cardplayer-magazines/65582-19-2/articles/15250-are-you-sage-getting-an-edge-in-heads-up-no-limit-hold-39-em (cited: January 15, 2010).

[41] Marv742, “Multi-player Flop Tables.”, http://games.cs.ualberta.ca/poker/Flop_Table/ (cited: January 15, 2010).

[42] Nicolai, Garrett, and Robert J. Hilderman, “No-Limit Texas Hold’em Poker Agents Created with Evolutionary Neural Networks” (Milan, Italy, 8 September, 2009), 2009 IEEE Symposium on Computational Intelligence and Games, http://www.ieee-cig.org/cig-2009/Proceedings/proceedings/papers/cig2009_018e.pdf.

[43] Félix, Dinis, Artificial Intelligence Techniques in Games with Incomplete Information: Opponent Modeling in Texas Hold’em. Faculdade de Engenharia da Universidade do Porto. Porto, March 2008. Master thesis.

[44] McNally, Patrick and Raffi, Zafar, “Opponent Modeling in Poker Using Machine Learning Techniques”, Northwestern University, 2008. Available online at http://www.cs.northwestern.edu/~ddowney/courses/349/projects/poker/Poker.pdf

[45] Wikipedia, the free encyclopedia, “Internet Bot”, Wikipedia.org http://en.wikipedia.org/wiki/Internet_bot (cited: January 15, 2010).

[46] “Bonus Bots Gaming Assistance Software,” http://www.bonusbots.com/ (cited: January 15, 2010).

[47] Mystery, Man Of, “How to Find a Working Pokerbot and Make it Win for You.” Poker Bot Blogspot http://pokerbotwin.blogspot.com/ (cited: January 15, 2010).

[48] “2010 Computer Poker Competition”, http://webdocs.cs.ualberta.ca/~pokert/ (cited: January 10, 2010).

[49] “Poker Bot Artificial Intelligence Resources”, http://spaz.ca/poker/ (cited: June 24, 2010).

[50] “Cactus Kev's Poker Hand Evaluator”, http://www.suffecool.net/poker/evaluator.html (cited: June 24, 2010).

[51] “Coding the Whell: Poker Hand Evaluator Roundup”, http://www.codingthewheel.com/archives/poker-hand-evaluator-roundup (cited: June 24, 2010).

[52] “Pokersource Poker-Eval”, http://pokersource.sourceforge.net/, (cited: June 24, 2010).

[53] “Paul Senzee on Software, Game Development and Technology”, http://www.senzee5.com/2007/01/7.html, (cited: June 25, 2010).

[54] “Foldem Hand Evaluator”, http://code.google.com/p/foldem/, (cited: June 25, 2010).

[55] B. Chen e J. Ankenman, The Mathematics of Poker, Conjelco, 2006.

[56] “Poker programmer”, “Poker for programmers blog”, http://pokerforprogrammers.blogspot.com/, (cited: June 25, 2010).

[57] “Poker Academy Pro – The Ultimate Poker Software”, available online at http://www.poker-academy.com, (cited: June 10, 2010).

[58] “Poker Stove – Poker Odds Calculator”, available online at http://www.pokerstove.com/pokerstove/features.php, (cited June 25, 2010).

[59] Usama Fayyad, Gregory Piateski-Shapiro, Padhraic Smyth. The KDD Process for Extracting Useful Knowledge from Volumes of Data. In: Communications of the ACM, pp.27-34, November 1996.

[60] M. Berry e G. Linoff. Data Mining Techniques for Marketing, Sales and Customer Support. New York: J. Wiley and Sons, 1997.

[61] Sholom M. Weiss, Nitin Indurkhya. Predictive Data Mining, San Francisco: Morgan Kaufmann Publishers, 1998.

[62] B. Tabachnick e L.S. Fidell. Using Multivariate Statistics. 3rd ed. New York: Harper Collins, 1996.

[63] Jiawei Han e Micheline Kamber. Data Mining Concepts and Techniques. New York: Academic Press, 2001.

[64] S. A. Nimeh, D. Nappa, and X., Nair, S. Wang, "A Comparison of Machine Learning Techniques for Phishing Detection," in Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit, vol. 269, Pittsburgh, 2007, pp. 60-69.

[65] S. Russell e P. Norvig, Artificial Intelligence: A Modern Approach, Pearson Education, 1998.

[66] M. Berry e G. Linoff. Data Mining Techniques for Marketing, Sales and Customer Support. New York: J. Wiley and Sons, 1997.

[67] Cybenko, G. 1989. Approximation by superpositions of a sigmoidal function *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4), 303–314.

[68] Pang-Nin Tan, Michael Steinback, and Vipin Kumar, Introduction to Data Mining.: Pearson Education, Inc, 2006.

[69] J. Platt: Fast Training of Support Vector Machines using Sequential Minimal Optimization. In B. Schoelkopf and C. Burges and A. Smola, editors, Advances in Kernel Methods - Support Vector Learning, 1998.

[70] Campbell, Colin. Introduction to Support Vector Machines, University of Bristol, January 2008, http://videolectures.net/epsrcws08_campbell_isvm/ (cited: June 23, 2010).

[71] Quinlan, J. R. C4.5: Programs for Machine Learning. Morgan Kaufmann Publishers, 1993.

[72] Liaw, Andy & Wiener, Matthew "Classification and Regression by randomForest" R News (2002) Vol. 2/3 p. 18

[73] Shi, Haijian. Best-first Decision Tree Learning Master Thesis, Department of Computer Science, The University of Waikato, Hamilton, New Zealand

[74] Mehmed Kantardzic. Data Mining. Concepts, Models, Methods, and Algorithms, Piscataway, NJ: IEEE Press, 2003.

[75] Ian H. Witten e Frank Eibe. Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations. 2nd ed. St. Louis: Morgan Kaufmann, 2005.

[76] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.

[77] *Mierswa, Ingo and Wurst, Michael and Klinkenberg, Ralf and Scholz, Martin and Euler, Timm:* YALE: Rapid Prototyping for Complex Data Mining Tasks, in Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-06), 2006.

[78] G. Broeck, K. Driessens, e J. Ramon, “Monte-Carlo Tree Search in Poker Using Expected Reward Distributions,” Proceedings of the 1st Asian Conference on Machine Learning: Advances in Machine Learning, Nanjing, China: Springer-Verlag, 2009, pp. 367-381.

[79] “Opentestbed”, http://code.google.com/p/opentestbed/, (cited: June 20).

[80] “Open Holdem”, http://code.google.com/p/openholdembot/, (cited: June 20).

[81] “Win Holdem”, http://www.winholdem.net/, (cited: June 20).

[82] Félix, D. and Reis, L. P. 2008. Opponent Modelling in Texas Hold'em Poker as the Key for Success. In *Proceeding of the 2008 Conference on ECAI 2008: 18th European Conference on Artificial intelligence* M. Ghallab, C. D. Spyropoulos, N. Fakotakis, and N. Avouris, Eds. Frontiers in Artificial Intelligence and Applications, vol. 178. IOS Press, Amsterdam, The Netherlands, 893-894.

[83] Felix, D. and Reis, L. P. 2008. An Experimental Approach to Online Opponent Modeling in Texas Hold'em Poker. In *Proceedings of the 19th Brazilian Symposium on Artificial intelligence: Advances in Artificial intelligence* (Savador, Brazil, October 26 - 30, 2008). G. Zaverucha and A. L. Costa, Eds. Lecture Notes

In Artificial Intelligence, vol. 5249. Springer-Verlag, Berlin, Heidelberg, 83-92. DOI= http://dx.doi.org/10.1007/978-3-540-88190-2_14

[84] "Out Floppped Poker Q & A – Obfuscated datamined hand histories", http://www.outflopped.com/questions/286/obfuscated-datamined-hand-histories, (cited: June 28)

[85] "HandHQ.Com", http://www.handhq.com/, (cited: June 28)

Bibliography

Appendix A – Glossary of Poker Terms

- **All-in.** To have one's entire stake committed to the current pot. Action continues toward a side pot, with the all-in player being eligible to win only the main pot.
- **All-in Equity.** The expected value income of a hand assuming the game will proceed to the showdown with no further betting (i.e., a fraction of the current pot, based on all possible future outcomes).
- **Bad Beat.** An unlucky loss. In particular, losing a game where the opponent probably should have folded, but instead got extremely lucky to win.
- **Bet.** To make the first wager of a betting round (compare raise).
- **Bet for Value.** To bet with the expectation of winning if called (compare bluff).
- **Big Bet.** The largest bet size in Limit poker (e.g., $20 in $10-$20 Hold'em).
- **Big Blind (sometimes called the Large Blind).** A forced bet made before the deal of the cards (e.g., $10 in $10-$20 Hold'em, posted by the second player to the left of the button).
- **Blind.** A forced bet made before the deal of the cards (see small blind and big blind).
- **Bluff .** To play a weak hand as though it were strong, with the expectation of losing if called (see also semi-bluff and pure bluff , compare bet for value).
- **Board (or Board Cards).** The community cards shared by all players.
- **Board Texture.** Classification of the type of board, such as having lots of high cards, or not having many draws (see dry).
- **Button.** The last player to act in each betting round in Texas Hold'em. Also called the dealer button, representing the person who would be the dealer in a home game.
- **Call.** To match the current level of betting. If the current level of betting is zero, the term check is preferred.
- **Cap.** (a) The maximum number of raises permitted in any single round of betting (typically four in Limit Hold'em, but occasionally unlimited). (b) (vt) To make the last permitted raise in the current betting round (e.g., after a bet, raise, and re-raise, a player caps the betting).

- **Check.** To decline to make the rst wager of a betting round (compare call).
- **Check-Raise.** To check on the first action, with the intention of raising in the same betting round after an opponent bets.
- **Community Cards.** The public cards shared by all players.
- **Connectors.** Two cards differing by one in rank, such as 7-6. More likely to make a straight than other combinations.
- **Dominated.** A Hold'em hand that has a greatly reduced chance of winning against another because one or both cards cannot make a useful pair (e.g., KQ is dominated by AK, AQ, AA, KK, and QQ, but not by AJ or JJ).
- **Draw.** A holding with high potential to make a strong hand, such as a straight draw or a flush draw (compare made hand).
- **Draw Potential.** The relative likelihood of a hand improving to be the best if it is currently behind.
- **Drawing Dead.** Playing a draw to a hand that will only lose, such as drawing to a flush when the opponent already holds a full house.
- **Drawing Hand.** A hand that has a good draw (compare made hand).
- **Dry.** Lacking possible draws or betting action, as in a dry board or a dry game.
- **Equity (or Pot Equity).** An estimate of the expected value income from a hand that accounts for future chance outcomes, and may or may not account for the effects of future betting (e.g., all-in equity).
- **Expected Value (EV) (also called mathematical expectation).** The average amount one expects to win in a given game situation, based on the payoffs for each possible random outcome.
- **Flop.** The first three community cards dealt in Hold'em, followed by the second betting round (compare board).
- **Fold.** To discard a hand instead of matching the outstanding bet, thereby losing any chance of winning the pot.
- **Fold Equity.** The equity gained by a player when an opponent folds. In particular, the positive equity gained despite the fact that the opponent's fold was entirely correct.
- **Forward Blinds.** The logical extension of blinds for heads-up (two-player) games, where the first player posts the small blind and the second player (button) posts the big blind (compare reverse blinds). (Both rules are seen in practice, with various casinos and online card rooms having different policies for multi-player games that have only two active players).
- **Free-Card Danger.** The risk associated with allowing an opponent to improve and win the pot without having to call a bet (in particular, when they would have folded).
- **Free-Card Raise.** To raise on the flop intending to check on the turn.
- **Game.** (a) A competitive activity in which players contend with each other according to a set of rules (in poker, a contest with two or more players). (b) A single instance of such an activity (in poker, from the initial dealing of the cards to the showdown, or until one player wins uncontested).

- **Game Theory.** Among serious poker players, game theory normally pertains to the optimal calling frequency (in response to a possible bluff), or the optimal bluffing frequency. Both depend only on the size of the bet in relation to the size of the pot.
- **Hand.** (a) A player's private cards (e.g., two hole cards in Hold'em). (b) One complete game of poker (see game (b)).
- **Heads-up.** A two-player (head-to-head) poker game.
- **Hole Card.** A private card in poker (Texas Hold'em, Omaha, 7-Stud, etc.).
- **Implied Odds.** (a) The pot odds based on the probable future size of the pot instead of the current size of the pot (positive or negative adjustments). (b) The extra money a strong hand stands to win in future betting rounds (compare reverse implied odds).
- **Kicker.** A side card, often deciding the winner when two hands are otherwise tied (e.g., a player holding Q-J when the board is Q-7-4 has top pair with a Jack kicker).
- **Large Blind (usually called the Big Blind).** A forced bet made before the deal of the cards (e.g., $10 in $10-$20 Hold'em, posted by the second player to the left of the button).
- **Loose Game.** A game having several loose players.
- **Loose Player.** A player who does not fold often (e.g., one who plays most hands at least to the op in Hold'em).
- **Made Hand.** A hand with a good chance of currently being the best, such as top pair on the op in Hold'em (compare draw).
- **Mixed Strategy.** Handling a particular type of situation in more than one way, such as to sometimes call, and sometimes raise.
- **Offsuit.** Two cards of different suits (also called unsuited, compare suited).
- **Open-Ended Draw.** A draw to a straight with eight cards to make the straight, such as 6-5 with a board of Q-7-4 in Hold'em.
- **Outs.** Cards that will improve a hand to a probable winner (compare draw).
- **Pocket Pair.** Two cards of the same rank, such as 6-6. More likely to make three of a kind than other combinations (see set).
- **Post-flop.** The actions after the flop in Texas Hold'em, including the turn and river cards interleaved with the three betting rounds, and ending with the showdown.
- **Pot.** The common pool of all collected wagers during a game.
- **Pot Equity (or simply Equity).** An estimate of the expected value income from a hand that accounts for future chance outcomes, and may or may not account for the effects of future betting (e.g., all-in equity).
- **Pot Odds.** The ratio of the size of the pot to the size of the outstanding bet, used to determine if a draw will have a positive expected value.
- **Pre-fop.** The first round of betting in Texas Hold'em before the flop, beginning with the posting of the blinds and the dealing of the private hole cards.
- **Pure bluff .** A bluff with a hand that can only win if the opponent folds (compare semi-bluff).
- **Pure Drawing Hand.** A weak hand that can only win by completing a draw, or by a successful bluff .

- **Raise.** To increase the current level of betting. If the current level of betting is zero, the term bet is preferred.
- **Raising for a Free-card.** To raise on the op intending to check on the turn.
- **Rake.** A portion of the pot withheld by the casino or host of a poker game, typically a percentage of the pot up to some maximum, such as 5% up to $3.
- **Re-raise.** To increase to the third level of betting after a bet and a raise.
- **Reverse Blinds.** A special rule sometimes used for heads-up (two-player) games, where the second player (button) posts the small blind and the first player posts the big blind (compare forward blinds). (Both rules are seen in practice, with various casinos and online card rooms having different policies for multi-player games that have only two active players).
- **Reverse Implied Odds.** The unaccounted (negative) money a mediocre hand stands to lose in future betting rounds (compare implied odds (b)).
- **River.** The fifth community card dealt in Hold'em, followed by the fourth (and final) betting round.
- **Semi-bluff .** A bluff when there are still cards to be dealt, with a hand that might be the best, or that has a reasonable chance of improving to the best if it is called (compare pure bluff).
- **Second pair.** Matching the second highest community card in Hold'em, such as having 7-6 with a board of Q-7-4.
- **Session.** A series of games, typically lasting several hours in length.
- **Set.** Three of a kind, formed with a pocket pair and one card of matching rank on the board. A very powerful and well-disguised hand (compare trips).
- **Short-handed Game.** A game with less than the full complement of players, such as a Texas Hold'em game with ve or fewer players.
- **Showdown.** The revealing of cards at the end of a game to determine the winner.
- **Side pot.** A second pot for the remaining active players after another player is all-in.
- **Slow-play.** To check or call a strong hand as though it were weak, with the intention of raising in a later betting round (compare smooth-call and checkraise).
- **Small Bet.** The smallest bet size in Limit poker (e.g., $10 in $10-$20 Hold'em).
- **Small Blind.** A forced bet made before the deal of the cards (e.g., $5 in $10-$20 Hold'em, posted by the first player to the left of the button).
- **Smooth-call.** To only call a bet instead of raising with a strong hand, for purposes of deception (as in a slow-play).
- **Suited.** Two cards of the same suit, such as both Hearts. More likely to make a flush than other combinations (compare offsuit or unsuited).
- **Table Image.** The general perception other players have of one's play.
- **Table Stakes.** A poker rule allowing a player who cannot match the outstanding bet to go all-in with his remaining money, and proceed to the showdown (also see side pot).
- **Texture of the Board.** Classification of the type of board, such as having lots of high cards, or not having many draws (see dry).

- **Tight Player.** A player who usually folds unless the situation is clearly profitable (e.g., one who folds most hands before the flop in Hold'em).
- **Time Charge.** A fee charged to the players in a poker game by a casino or other host of the game, typically collected once every 30 minutes.
- **Top Pair.** Matching the highest community card in Hold'em, such as having Q-J with a board of Q-7-4.
- **Trap.** To play a strong hand as though it were weak, hoping to lure a weaker hand into betting. Usually a check-raise or a slow-play.
- **Trips.** Three of a kind, formed with one hole card and two cards of matching rank on the board. A strong hand, but not well-disguised (compare set).
- **Turn.** The fourth community card dealt in Hold'em, followed by the third betting round.
- **Unsuited.** Two cards of different suits (also called offsuit, compare suited).
- **Value Bet.** To bet with the expectation of winning if called (compare bluff).
- **Wild Game.** A game with a lot of raising and re-raising. Also called an action game.

Appendix A – Glossary of Poker Terms

Appendix B - HoldemML XML Schema

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
 <xs:element name="HoldemML">
  <xs:complexType>
   <xs:sequence>
    <xs:element name="Games" minOccurs="1" maxOccurs="1">
     <xs:complexType>
      <xs:sequence>
       <xs:element name="Game" minOccurs="0" maxOccurs="unbounded">
        <xs:complexType>
         <xs:sequence>
          <xs:element name="Players" minOccurs="1" maxOccurs="1">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="Player" type="player" minOccurs="1"
              maxOccurs="unbounded"/>
             <xs:element name="Dealer" type="player" minOccurs="1" maxOccurs="1"/>
            </xs:sequence>
           </xs:complexType>
          </xs:element>
          <xs:element name="PreFlop" minOccurs="1" maxOccurs="1">
           <xs:complexType>
            <xs:sequence>
             <xs:element name="SmallBlindBet" type="bet-action" minOccurs="1"
              maxOccurs="1"/>
             <xs:element name="BigBlindBet" type="bet-action" minOccurs="1"
              maxOccurs="1"/>
             <xs:choice minOccurs="1" maxOccurs="unbounded">
```

```
            <xs:element name="Bet" type="bet-action"/>
            <xs:element name="Call" type="bet-action" />
            <xs:element name="Raise" type="bet-action"/>
            <xs:element name="Fold" type="action"/>
            <xs:element name="Check" type="action"/>
            <xs:element name="All-In" type="bet-action"/>
           </xs:choice>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
        <xs:element name="Flop" minOccurs="0" maxOccurs="1"
         type ="three_cards_round"/>
        <xs:element name="Turn" minOccurs="0" maxOccurs="1" type="one_card_round"/>
        <xs:element name="River" minOccurs="0" maxOccurs="1" type="one_card_round"/>
        <xs:element name="ShowDown" minOccurs="1" maxOccurs="1">
         <xs:complexType>
          <xs:sequence>
           <xs:choice maxOccurs="unbounded" minOccurs="0">
       <xs:element name="Muck" minOccurs="0" maxOccurs="unbounded"
         type="action"/>
            <xs:element name="Show" minOccurs="0" maxOccurs="unbounded"
              type="showCards"/>
            <xs:element name="Collects" type="bet-action" minOccurs="1"
             maxOccurs="unbounded"/>
           </xs:choice>
          </xs:sequence>
         </xs:complexType>
        </xs:element>
       </xs:sequence>
       <xs:attribute name="id" type="xs:string" use="required"/>
       <xs:attribute name="big-blind" type="xs:float" use="required"/>
       <xs:attribute name="small-blind" type="xs:float" use="required"/>
       <xs:attribute name="date" type="xs:date" use="required"/>
       <xs:attribute name="table" type="xs:string" use="required"/>
      </xs:complexType>
      <xs:key name="PlayerNameIsKey">
       <xs:selector xpath="Players/*"></xs:selector>
       <xs:field xpath="Name"></xs:field>
      </xs:key>
      <xs:keyref refer="PlayerNameIsKey" name="PlayerNameRef">
       <xs:selector xpath="PreFlop/*|Flop/*|Turn/*|River/*|ShowDown/*"></xs:selector>
       <xs:field xpath="@player"></xs:field>
      </xs:keyref>
     </xs:element> <!-- End Game -->
```

```
        </xs:sequence>
       </xs:complexType>
     </xs:element> <!-- End Games -->
   </xs:sequence>
  </xs:complexType>
</xs:element> <!-- End HoldemML -->

<xs:complexType name="player">
  <xs:sequence>
    <xs:element name="Name" minOccurs="1" maxOccurs="1">
      <xs:simpleType>
        <xs:restriction base="xs:string">
          <xs:minLength value="1"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
    <xs:element name="ChipCount" minOccurs="1" maxOccurs="1">
      <xs:simpleType>
        <xs:restriction base="xs:float">
          <xs:minInclusive value="0"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:element>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="three_cards_round">
  <xs:sequence>
    <xs:element name="CommunityCards" type="three_cards" minOccurs="1" maxOccurs="1"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element name="Bet" type="bet-action"/>
      <xs:element name="Call" type="bet-action" />
      <xs:element name="Raise" type="bet-action"/>
      <xs:element name="Fold" type="action"/>
      <xs:element name="Check" type="action"/>
      <xs:element name="All-In" type="bet-action"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="one_card_round">
  <xs:sequence>
    <xs:element name="CommunityCards" type="one_card" minOccurs="1" maxOccurs="1"/>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
```

```
      <xs:element name="Bet" type="bet-action"/>
      <xs:element name="Call" type="bet-action" />
      <xs:element name="Raise" type="bet-action"/>
      <xs:element name="Fold" type="action"/>
      <xs:element name="Check" type="action"/>
      <xs:element name="All-In" type="bet-action"/>
    </xs:choice>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="three_cards">
  <xs:sequence>
    <xs:element name="Card" type="card" minOccurs="3" maxOccurs="3"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="one_card">
  <xs:sequence>
    <xs:element name="Card" type="card" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="two_cards">
  <xs:sequence>
    <xs:element name="Card" type="card" minOccurs="1" maxOccurs="1"/>
  </xs:sequence>
</xs:complexType>

<xs:complexType name="card">
  <xs:attribute name="rank" type="card-rank" use="required"/>
  <xs:attribute name="suit" type="card-suit" use="required"/>
</xs:complexType>

<xs:simpleType name="card-suit">
  <xs:restriction base="xs:string">
    <xs:enumeration value="S"/>
    <xs:enumeration value="H"/>
    <xs:enumeration value="D"/>
    <xs:enumeration value="C"/>
  </xs:restriction>
</xs:simpleType>

<xs:complexType name="showCards">
  <xs:sequence>
```

```
    <xs:element name="Card" type="card" minOccurs="1" maxOccurs="2"/>
   </xs:sequence>
   <xs:attribute name="player" type="xs:string" use="required"/>
  </xs:complexType>

  <xs:simpleType name="card-rank">
   <xs:restriction base="xs:string">
    <xs:enumeration value="A"/>
    <xs:enumeration value="K"/>
    <xs:enumeration value="Q"/>
    <xs:enumeration value="J"/>
    <xs:enumeration value="10"/>
    <xs:enumeration value="9"/>
    <xs:enumeration value="8"/>
    <xs:enumeration value="7"/>
    <xs:enumeration value="6"/>
    <xs:enumeration value="5"/>
    <xs:enumeration value="4"/>
    <xs:enumeration value="3"/>
    <xs:enumeration value="2"/>
   </xs:restriction>
  </xs:simpleType>

  <xs:complexType name="bet-action">
   <xs:attribute name="player" type="xs:string" use="required"/>
   <xs:attribute name="value" type="xs:float" use="required"/>
  </xs:complexType>

  <xs:complexType name="action">
   <xs:attribute name="player" type="xs:string" use="required"/>
  </xs:complexType>

</xs:schema>
```