Using machine learning to detect bots in World of Warcraft

By Ian D. Stevens

A thesis submitted for the degree of Bachelor of Information Technology (Honours) School of Computer and Information Science University of South Australia

Principle Supervisor: Wolfgang Mayer
Declaration

I declare that this thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge it does not contain any materials previously published or written by another person except where due reference is made in the text.

Signature of Author: .................................................................
Ian David Stevens

Cape Town
24th October 2016
Abstract

A game bot is an automated program that takes control of an avatar in a game to perform certain tasks. These game bots ruin the various aspects of the game for people who are playing the game as it was intended to be played. Most of the game companies state they do not allow botting in their game and it violates their user policy agreement.

There are many different systems in place that are design to classify character in a game as a human or a bot, however, these systems are far from perfect, and they can have adverse effects on the game they were trying to improve by increasing latency, ruining the immersion of the game, or invading the players privacy.

This paper proposes a bot detection method that avoids the aforementioned short falls of current systems. We look at various factors of game play in typical tasks and how bots and humans differ in executing these tasks using scientific analysis. Based on these factors we construct a classifier that can learn to tell the difference between a human and bot. In this paper we aim to show that by carefully observing these human traits we can design a classifier that can successfully detect bots while not interfering in other areas of the game. Using the popular massively multiplayer online game World of Warcraft we tested our classifier with real human test cases and popular bots used in the game. The results showed our classifier achieved a 87.65% Accuracy rate.
Contents

1 Introduction .......................................................... 1
  1.1 Background ...................................................... 1
  1.2 Plan of Development .......................................... 2
  1.3 Project Objectives .............................................. 4

2 The Economy of Gaming ............................................ 5

3 The History of Gold Farming ...................................... 8

4 Literature Review .................................................. 10
  4.1 Looking at the Bots .............................................. 10
  4.2 Where it Happens ............................................... 11
  4.3 Common Bot Detection Methods ............................... 13
    4.3.1 CAPTCHA Analysis ........................................ 13
    4.3.2 Traffic Analysis ......................................... 14
    4.3.3 User Behaviour Analysis ................................ 15
    4.3.4 Moving Path Analysis .................................... 18
    4.3.5 Human Observational Proofs ............................ 19
  4.4 Literature review conclusion .................................. 20

5 Machine Learning and Data Mining ............................... 22
  5.1 Major Types of Machine Learning ............................ 23
    5.1.1 Supervised Learning ..................................... 23
    5.1.2 Unsupervised learning .................................. 24
  5.2 General Problems Data Mining Algorithms Solve ........... 24
  5.3 Top data mining algorithms ................................... 25
    5.3.1 C4.5 algorithm ........................................... 25
    5.3.2 The k-means algorithm ................................... 27
5.3.3 Support vector machines (SVM) ........................................... 28
5.3.4 The EM algorithm ......................................................... 28
5.3.5 AdaBoost ................................................................. 30
5.3.6 kNN ............................................................... 30
5.3.7 Naïve Bayes ............................................................... 32
5.3.8 CART ................................................................. 32
5.3.9 Artificial neural network (ANN) ................................. 33

6 Methodology ................................................................. 34
  6.1 Research Questions ......................................................... 35
  6.2 Conditions for the experiment ........................................ 35
    6.2.1 The Location ......................................................... 35
    6.2.2 The WoW server ...................................................... 36
    6.2.3 Player Data Collectoin .............................................. 36
    6.2.4 The bot programs ...................................................... 37
  6.3 Bot data collection ......................................................... 37
    6.3.1 Data pre-processing ................................................ 39
    6.3.2 Classifier architecture and training ............................ 40

7 Results ................................................................. 43
  7.1 Calculating the information gain for each attribute .......... 44
  7.2 First Test .............................................................. 44
  7.3 Second test .............................................................. 45
  7.4 Missing attribute test .................................................. 46
  7.5 Bot profile outliers ..................................................... 47
  7.6 Extra attribute test ..................................................... 48

8 Discussion ............................................................. 52

9 Future Work .......................................................... 55

10 Conclusion .......................................................... 57

11 References .......................................................... 59
Nomenclature

- MMORPG, Massively Multiplayer Online Role Playing Game
- MMO, Massively Multiplayer Online
- WoW, World of Warcraft
- RMT, Real Money Transfers
- CAPTCHA, Completely Automated Public Turing test to tell Computers and Humans Apart
- HOP, Human Observation Proofs
- EI, emerging images
- DCG, Dynamic Cognitive Games
- HMMs, Hidden Markov Models
- AI, Artificial Inteligence
- SVM, Support Vector Machines
- EM, Expectation-Maximization
- KNN, K-Nearest Neighbour
- CART, Classification And Regression Tree
- ANN, Artificial Neural Network
Chapter 1

Introduction

1.1 Background

A game bot is an automated program that takes control of an avatar in a game to perform certain tasks. Bots are designed to level up characters, gather items in games, and/or give gamers an unfair edge by aiding in a function of game play. These bots cause various problems in the game and can ruin the enjoyment of the game for non botting players.

When players use game bots to level up their characters, they are misrepresenting their skill level. These players join high level raids or dungeons but do not perform to their expected skill level. Players that level up their characters properly get annoyed and angry for failing an in game challenge due to being paired with players that do not have the skill level one would expect when considering the character level they have achieved. Bots that are designed to gather items normally end up gathering in the same area farming a specific item.

Actual players find it hard to farm large amounts of these items because the bots farm all the spawn points faster and more efficiently than human players can. Some bots farm large amounts of a single item and then sell them in the in-game auction house for a low price, thus damaging the game economy.
This lowers the values of that specific item, making it a lot easier than it should be for players to acquire. Players that farm those items properly can no longer get the expected selling price for the item.

Some bots are aimed to perform a task while a human is playing, aim bots in first person shooters is a common example as they auto-focus a player’s crosshairs on to an enemy players or unfriendly Non Player Characters, consequentially eliminating the skill component required to be good at these games. These bots make it hard for legitimate player to compete. If players are competing against a player that seem unbeatable it discourages the normal player.

Game companies today have designed various methods to identify bots form legitimate players. When a player is discovered to be botting, various disciplinary actions can be taken against their account. Most typically botting results in banning of the associated account. Bot detection algorithms need to be able to detect a high number of bots while not misclassifying any legitimate players as a bot and taking action against actually innocent players. Most bot detection programs implemented in high end games achieve this but normally at the expense to another factor of the gaming experience. Bot detection methods implemented in games can: slow down the game, ruin the immersion of the game, take up processing power, or invade the player’s privacy. Many of the bot detection programs are easy to counter, as soon as they comes out, a new patch for the bot program is released to counter it and dishonest players can carry on cheating undetected.

1.2 Plan of Development

We commence this paper with a bit of background, starting with the economy of gaming, which looks at how much money is involved in the world of gaming and what motivates people go to such extremes to farm items. It discusses the trading of virtual items for real money and how much some of these items are worth.

The history of gold farming looks at the creation of the virtual economy and how it came to be. It examines previously used systems to farm items
on a large scale.

The literature review looks at a wide range of different techniques proposed for bot detection. It discusses both the merits and the pitfalls of these techniques and analyse which the best possible solution for bot detection is.

Machine learning and data mining takes a looks at the history of machine learning and what is involved in the process. It examines the top methods used in data mining that could be applied to our classifier.

The methodology describes a method of bot detection in the popular massively multiplayer online role playing game (MMORPG), World of Warcraft (WoW), created by Blizzard Entertainment, that will avoid the traditional shortfalls common to game bot detection methods. The proposed method looks at gathering a small amount of data from the game so as to avoid unnecessarily adding strain to the system. The information gathered is used to identify different factors about a player’s farming style. These different factors are selected according to how difficult it would be for a bot to replicate. Using all these different factors the author creates a machine learning classifier to distinguish between bots and legitimate players.

The results look at how well the described method can be used to classify bots. It looks at various factors such as true positives and false negatives, it also discuss the effect of adding more test cases to train the classifier before testing it. We look at using less factors, which means less strain on the system, and see its impact on the results of the experiment. It examines how well the classifier handles outliers, i.e. bot profiles set up in an illogical or irregular way.

We look into examining potential future work that could be done to improve the described method and test.

In our conclusion we answer the research question purposed in this paper.
1.3 Project Objectives

This paper aims to fundamentally answer the following research questions:

- Can bots be identified to a sufficient level by using efficiency factors related to gold farming?
- Can bots be identified to a sufficient level by using fewer efficiency factors related to gold farming?
- How well does the proposed method deal with bot profile outliers, can it identify a bot that is setup in an illogical way and detrimental to the farming of items?
Chapter 2

The Economy of Gaming

Many people both developers and players alike are not convinced botting is not a pressing and serious issue. It is quite likely that this is due to a lack of understanding in the amount of money and the economy involved in gaming.

In recent years the world of gaming has taken a turn towards making money through playing games. The public has seen the rise of professional gamers, people who have taken playing games and turned them into a very lucrative profession [1]. The professional gaming scene is becoming bigger and bigger. The top professional gamers earn salaries close to that of professional sports men. The highest earning e-sport player is Peter Dager a DOTA 2 player and captain of the team Evil Geniuses whose total earnings come to $2,178,940.76 (US) [1]. The majority of this money came from his team’s recent win of the 2015 DOTA 2 international, held by Valve. This tournament had the highest prize pool of any e-sport competition with the winning team getting $6,634,661 and the finals being shown on ESPN [2]. Although DOTA 2’s International has the biggest prize pool, its viewers (4.5 million) are not close to that of Riot’s League of Legends World Championship, which had 36 million unique viewers [3].
Professional or competitive gaming is not the only medium through which players turn gaming into a profitable pastime. Many gamers have inventively turned to selling virtual items for real money or real money transfers (RMT). Arguably the biggest sale in video game history for virtual items occurred in the Entropia Universe, a Sci-Fi massively multiplayer online role playing game (MMORPG) [4]. This game has some of the most expensive virtual items and many of players have made money by selling the items they have acquired through playing the game. One individual who stands out is Jon Jacobs who sold “Club Neverdie” in 2010 for $635,000(US). He sold it off in sections with the biggest individual sale coming to $335,000 [5].

There are two opposing opinions on the buying and selling of virtual items in online games. One is that it is acceptable, and game companies seek ways to help players do it safely and efficiently. Much of the time, however, money is exchanged but the virtual item is not. Game companies that support RMT provide a platform to minimize the chances of this happening. The other opinion is that involving real-world currency in the gaming world can have negative effects on the game immersion and economy. Most massively multiplayer online games (MMOs) in Japan prohibit RMT. It can cause inflation of certain items and make it harder for the typical player to acquire, or conversely, detract from in-game achievements that can now be acquired with money rather than skill. RTM players also camp out areas with high value items and steal other players’ items, in order to have highly marketable and valuable items to sell [6].
In the past, game operators would look for players associated with suspicious activities, such as having multiple accounts or reports from other players that pointed to RMT. If a player was suspected of RMT the game operators would manually observe him to make sure their suspicions were correct, because if they were found to be guilty their account would be banned. This approach was very time consuming for the game staff. Itsuki et al. looked at developing a more efficient approach to minimize the total number of suspect for RTM. This approach involved mining through the log data for the game checking for:

- Total action count: Total number of action records
- Activity time: The time that the player has been preforming a task
- Total chat count: The amount the player speaks to another player
- Total currency handled: the amount of currency the player accumulated

Looking at the above factors Itsuki et al. managed to narrow down the list of suspicious suspects [6]. With RMT generating so much money and becoming such a large factor in the gaming society a lot of players would spend countless hours just collecting in game item to sell.
Chapter 3

The History of Gold Farming

Gold farming is a common practice in MMO; it describes the action of acquiring in-game currency and then later selling it for real-world money. Richard Heeks identified three major events that he attributed the start of gold farming to:

- “The launch of Ultima Online, which became the first true massively-multiplayer online game.
- The launch of eBay, which provided a low-cost mechanism for the offer and sale of virtual items.
- The Asian currency crisis in which Asian governments sought to spend their way out of the crisis by investing heavily in broadband infrastructure. Some of those who became unemployed set up new businesses such as PC kiosks in which games could be played, and others among the unemployed turned to games playing to fill their empty hours. As a result, a strong games culture including gaming skills and entrepreneurship took root in East Asia. [7]”

These three events were a catalyst for the beginning of the virtual economy. The origin of commercialized gold farming is in South Korean, where cyber-cafés were being turned into operations for gold farming [8].
The exchange of virtual world goods was slowly becoming a capitalist economy all of its own. At the beginning of the 21st century it exhibited typical late-phase features of capitalist society.

- **Wage labour**: where entrepreneurs would pay hired workers to farm gold on their behalf.

- **Offshoring**: where they migrated the labour cost to low income areas (in this case mainly East Asia and China) to cut production cost.

- **Automation**: The cutting of time and financial cost by using bots that can imitate the actions of real players and can be used to farm virtual items [4].

As with any system that indicates the potential to make money, some untoward actions we bound to happen. Liu Dali, a prisoner at the Jixi labour camp was forced to play online games for the benefit of the guards. He stated that, “the prison bosses made more money forcing inmates to play games than they do forcing people to do manual labour”. The 300 prisoners were forced to play games in 12 hour shifts and would earn 5,000-6,000rmb [£470-570] a day. The prisoners never saw any of the money they earned and the computers were never turned off [9]. The virtual market has become such a lucrative business that different countries have started to impose tax on the buying and selling of virtual items [10].

As one can see, gold farming and professional gaming is a sizeable and lucrative market, and a large part of this is due to the creation of gaming bots. Many bots can be set to act in different ways, follow a certain route, and gather different types of resources, respond to a player talking to you, and to log off after a certain amount of time as not to raise suspicion. Some games like Rosh Online encourage the use of game bots and allow the users 6 hours of botting per day, but other games like World of Warcraft (WoW) are against it because of the various problems they cause that were discussed earlier. Bot makers also implement different strategies when designing their bots depending on the platform being targeted.
Chapter 4

Literature Review

4.1 Looking at the Bots

A game bot is an automated program that takes control of an avatar in a game to performs certain tasks. They are normally used to perform mindless repetitive task that the human player does not want to do or do not have time to do. They can also be used to enhance a certain aspect of game play such as aim bots in a first person shooter. Bots are effective as they can be playing the game 24/7. Most game companies do not allow bots, though many players still use them. Companies have developed different strategies to detect bots in their games, however, this has caused bot to evolve and to be programmed to avoid detection. Some of their features commonly found in bots include: emulation of game packets perfectly, evasion for CAPTCHA authentication, automated chat responses for evading social interaction monitoring, and OCR recognition [11].

There are two general approaches when creating a bot:
Custom game client: The bot companies create their own custom game client that is designed to interact with the server the way the normal game client would. This method takes a lot of work as the bot designers have to reverse engineer the real game client so it interacts with the game sever exactly like the original would. As games are become more complex this form of bot development is being phased out of high end MMORPGs but it is a good approach to creating bots for mobile games such as Clash of Clans [12].
Manipulating game interface: This method involves sending mouse and keyboard inputs to the game the way a normal human player would. This approach is used more often in today’s MMORPGs as they are easier to develop. To understand the current state of the game, the bots scan pixel values of larger areas of the screen and use text recognition software to understand what is happening inside the game [12]. Just as game bots differ so does the approach used to detect them and where the detection takes place.

4.2 Where it Happens

The bot detection process happens at different parts of the game’s architecture. The main concern is how to detect bots without having a negative effect on that part of the game’s architecture. Kang et al. identified three areas where bot detection can take place as seen in Table 1: client-side, network-side, and server-side [13].

Client-side involves detecting the bot by analysing the player’s computer. This can involve but is not limited to, looking at what programs are running on the player’s computer, monitor key strokes and mouse movement, and adding elements into the game that will not show on the monitor but a bot will recognise and interact with. An example of this is Blizzard’s warden system for WoW. It is very effective and in May 2015 Blizzard banned more than 100,000 WoW accounts which they claimed to be using bots and made HonorBuddy (A popular botting program for WoW) close up its authentication system [14]. The Warden System comes with its own problems as people are concerned it violates their privacy. In Blizzard Entertainment’s end user policy, it states that their “client software may transmit certain geographic information of information regarding your computer (capabilities, game data Processing, etc)” [15] and this information is used to tell if you are running botting software on your computer. This added to the fact that Blizzard will not disclose how the system works or what information it is looking for/sending has made many a WoW players concerned about their privacy. If Blizzard did disclose how Warden works then bot companies can work out how to construct their bots in such a way to go unnoticed.
Network-side detection looks at the traffic between the player and the server. There are two popular methods to achieve bot detection on the network-side. The first is traffic monitoring, this normally involves learning specific traits of traffic that humans/bots exhibit while playing a game. It then builds a model to classify the suspected players as either a human or a bot based on their traffic tendencies [13, 16, 17]. The second is network protocol change, an example is L2s-Guard, it protects against reading, editing, and spoofing network packets, stopping third party programs from accessing your game information and controlling your avatar [13, 18].

Server-side detection involves looking at the behaviour of the avatar by monitoring their inputs that are stored on the server. Detecting bots through server-side detection has an advantage because it does not interfere with the player in any way, be it on his computer or his network connection. It normally looks for actions that are inconsistent with the way a human plays. Some of the more common things to look for would be:

- Amount of time spent in the game, if the player has been active for long periods of time this is a common indicator that a bot is controlling the avatar
- Checking the players inventory, if the player is consistently selling a certain type of item he may be using a bot to gather resources for him
- The avatar does not respond to other players when they attempt to communicate

These problems are easily overcome as bot designers implement settings to go unnoticed, log off after a specified time, implement a built in chat system or simply log off as soon as a game’s GM tries to talk to them.

To detect bot through server-side detection research has been done in checking for human characteristics that are harder to emulate. We talk about some of these human characteristics later in the paper. There have been a wide variety of different approaches when it comes to detecting game bots and a lot of academic research has been done on the subject. All of these approaches seem to suffer from one shortfall or another though.
4.3 Common Bot Detection Methods

Ah Kang et al. then classified common bot detection algorithms into 5 main
categories as seen in Table 3: Completely automated public Turing test to tell
computers and humans apart (CAPTCHA) analysis, Traffic analysis, user
behaviour analysis, Moving path analysis, and Human observation proofs
(HOP) [13].

4.3.1 CAPTCHA Analysis

CAPTCHA analysis involves presenting the player with a test only a human
can solve to see if the player is a bot or a real person. Some common types
of CAPTCHA test are:

- Reading a password displayed as a cluttered image;
- Identifying complex shapes;
- Rendering spatial text images from 3D models;
- Quizzing visual or audio puzzles or trivia questions;
- Matching common themes for a set of related images;
- Navigating virtual reality in a 3D world;
- Using media files collected from the real word, particularly the web
  naturally; and
- Incorporating an implicit test into the web page navigation system [10].

A common problem in CAPTCHA analysis is that it diminishes the immer-
sion of the game as users have to stop playing the game in order to take
the test [13, 19, 20]. This type of bot detection test works best when it
needs to be a once off test such as making an account for a social networking
site. It does not work well when it has to be continually checking for a bot,
Be in online games. There are also two common methods to overcoming a
CAPTCHA test:

1. Outsourcing the problem to a person (relay attack): The bot sends
   static snapshots to the human solver and receives and copies their an-
   swer. Another way is the bot utilises streaming software to send the
game frames/interactions to the solver and then receives their answer.
2. Automated attacks using image processing and machine learning. Many CAPTCHA tests ask you to identify all images with a common themes/object. Using image processing and machine learning algorithms they identify images with common themes/objects [21, 22].

Gao et al. attempted to solve both these problems by amalgamating emerging images (EI) and dynamic cognitive games (DCG). EI is a type of CAPTCHA test that typically uses moving images making it hard to be recognised by machine learning programs; it is known to be resistant to automated attacks [21]. DCG is a type of CAPTCHA test that “challenge the user to perform a game-like cognitive task interacting with a series of dynamic images.” DCG it is known to be resistant to relay attacks. This is because when a human plays a flash based game rendered on their computer they are more proficient than if a bot had to stream a game to solver, this is because of the latency of the communication channel between the bot and the solver [21].

Gao et al. also incorporated “pseudo 3D object rotation, incomplete object contour and tiling background, to reduce the information exposure through the superimposition of consecutive frames, and a fast frame rate to resist the relay attack based on streaming.” Their technique was resistant to automated attacks and made it difficult to perform relay attacks and easy for them to be detected [22].

4.3.2 Traffic Analysis

Traffic analysis monitors various factors (release time of client commands, magnitude of traffic burstiness, sensitivity to different networks conditions, etc) of the behaviour of the traffic coming from the user’s computer [17, 23]. Through these factors it can identified whether the game is being played by a bot or person. Some traffic analysis techniques (probing or chatting) suffer from a major flaw. They generate additional packets which can lead to slowing the connection down thus making the game less enjoyable for the user [8, 15]. Due to this flaw in traffic analysis, it does not lend itself well to bot detection in online games.
Yuanchao Lu et al. trained hidden Markov models (HMMs) with network samples of game bots and tested avatars of interest against the common behaviour found with the HMMs. Yuanchao Lu et al. restricted themselves to time-based traffic analysis. By doing so the researchers did not interfere with the packet timing (the player’s connection speed). To protect games privacy “traffic is encrypted end-to-end or by directing game packets through anonymity networks such as Tor [17]”. This method obtains a detection rate higher than 0.8 and a false negative, false positive rate of less than 0.2 using less than 2,000 packets to train the HMMs. This method of bot detection is very flexible as it can not only be applied to different types of MMOs but possibly to detect bots used in other areas such as click fraud. Bot makers may be able to overcome this detection method by delaying game packets randomly or to match human behaviour.

4.3.3 User Behaviour Analysis

There has been a lot of research done in this field as there are countless human characteristics that researches can look for when trying to identify a bot in a game. Most research considers human characteristics that are hard for a program to mimic. Features that are considered AI-complete (implying that developing these features for a bot would be equivalent to that of solving the central artificial intelligence problem) would be a first choice when looking for bots in online games, unfortunately however, most game features in game would not be considered AI complete. Nonetheless, there are still player-game interactions that are notably difficult for bots to reproduce.

Interacting with other players is a big part of playing a MMO and although bots come with chatting responses to evade GM monitoring they cannot mimic the complexity of how humans talk to each other. Kang et al. worked on detecting game bots by looking at the speech patterns mainly based on content and the repetitiveness of their speech [24].
Talking is just one part of socialising in online games, players typically like to form groups or parties to perform certain activities together. The reasons why human form parties is usually to complete a quest that requires more people or is too hard for one person to accomplish. Bots on the other hand form parties to gather resources, the bots normally form parties of two players and one gathers the items while the other acts as a bodyguard. Kang et al. looked at the party-play log to identify bots in MMORPGS. Looking at the duration parties stayed together Kang et al. identified potential bot parties as they tend to stay together longer, to farm more. After that the suspicious party’s actions were looked at to see if there was a high rate of gathering actions compared to total actions [13].

Chung et al. took into account that bots can be programmed to accomplish different tasks. Also importantly identified were the typical behavioural features that a player might exhibit while playing the game as shown in Table 2. These features build up the basic categories within a game: battle, collect, and move. Categorising a player based on their different play styles made bot detection much easier. The test focused on the difference between humans and bots and not on the fact that different players have different ways of playing. Due to this fact the test done to identify bots in each group were simpler, putting less strain on the system and could sample the data at low resolution [25].
### Table 1. Description of behavioral features

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hunting</td>
<td>Accumulated number of hunted NPCs for a sampling interval</td>
</tr>
<tr>
<td>Attack</td>
<td>Accumulated attack count for a sampling interval</td>
</tr>
<tr>
<td>Hit</td>
<td>Accumulated hit (successful attack) count for a sampling interval</td>
</tr>
<tr>
<td>Defense</td>
<td>Accumulated defense count for five minutes</td>
</tr>
<tr>
<td>Avoidance</td>
<td>Accumulated avoidance (successful defense) count for a sampling interval</td>
</tr>
<tr>
<td>Recovery</td>
<td>Accumulated number of healing potion usages for a sampling interval</td>
</tr>
<tr>
<td>Item</td>
<td>Number of items at the time of sampling</td>
</tr>
<tr>
<td>Collection</td>
<td>Accumulated number of collected items for a sampling interval</td>
</tr>
<tr>
<td>Drop</td>
<td>Accumulated number of dropped items for a sampling interval</td>
</tr>
<tr>
<td>X</td>
<td>Coordinate X at the time of sampling</td>
</tr>
<tr>
<td>Y</td>
<td>Coordinate Y at the time of sampling</td>
</tr>
<tr>
<td>Portal</td>
<td>Accumulated number of portal item usage count for a sampling interval</td>
</tr>
</tbody>
</table>

Figure 4.1: Description of Behavioural Features [13]
4.3.4 Moving Path Analysis

Moving path analysis involves recording the path taking of an avatar and based off certain attributes classify it as a bot or a human. Human movement is often random while a bot is determined by a set of way points and will only differ from this path to attack a monster or to farm an item. Mitterhofer et al. looked at the path taken by a number of different players and transformed that data into intermediate results such as waypoints or path segments. It was shown that a detection metric could be developed based on the amount of time an avatar visits a way point or takes the same path segment [26].

Chen et al. took a different approach and identified four features based off how an avatar moves:

1. Ideal activity, the amount of time a player stays in one place.
2. Pace, the speed of the avatar in the game. A player can move at different speed with in the game
3. Path,
   - Linger, “For an avatar at \((x, y)\) at time \(t\), if its distance from \((x, y)\) was always less than \(d\) during the period \((t, t + p)\), we say that the avatar was lingering during \((t, t+p)\), given the parameters \((d, p)\).”
   - Smoothness, “...determines whether an avatar moves in straight or zig-zag patterns.”
   - Detour “If an avatar is at \((x_1, y_1)\) at time \(t_1\) and at \((x_2, y_2)\) at time \(t_2\), we compute the detour by dividing the length of the movement by the effective offset of an avatar during the period \((t_1, t_2)\).”
4. Turn, the change of direction the avatar undergoes

Looking at these factors Chen et al. managed to classify bot in Quake 2 using kNN and SVM algorithms [27].
4.3.5 Human Observational Proofs

As we have discussed a common way to design a bot is through manipulating the game interface. It is only natural to look if there is a difference between the input a bot provides and the input a human player provides. Barik et al. developed a concept called spatial game signatures, where they look at if they could identify a bot by the position of mouse clicks on the screen. They found that not all mouse clicks are normally distributed about the centre of the screen. Using this information they were able to detect bots in social games [28].

Dominguez et al. took it a step further and mapped different cognitive processes to different ways people used their input devices. This map between cognitive thought and input supplied would be extremely hard for bot programmers to install. This process is still in early development phase and has not been tested/tried in a real world scenario; however the results showed that different cognitive processes involved in solving a task have physical manifestations that can be detected through low-level input pattern-mining [29].
4.4 Literature review conclusion

There have been many different bot detection methods created and they all come with their own pros and cons, as is shown in Table 2 below.

<table>
<thead>
<tr>
<th>Category</th>
<th>Adapted method</th>
<th>Definition/key papers</th>
<th>Key Idea</th>
<th>Merits/demerits</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>- Demeint: reducing immersion of players in online game</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Low feasibility</td>
</tr>
<tr>
<td>Network side detection</td>
<td>Traffic analysis</td>
<td>Detection method based on network traffic analysis [9,9]</td>
<td>- Command packets timing analysis</td>
<td>- Merit: high utilization of the other algorithms (like decision tree)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Traffic explosiveness analysis</td>
<td>- Demeint: low accuracy rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Networks response analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Data length analysis</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Traffic interval time analysis</td>
<td></td>
</tr>
<tr>
<td>Server side detection</td>
<td>User behavior analysis</td>
<td>Detection method based on user behavior pattern in game play [10–15]</td>
<td>- Idle time analysis</td>
<td>- Merit: high accuracy rate, high detection rate, high availability</td>
</tr>
<tr>
<td></td>
<td>Moving path analysis</td>
<td>Detection method based on patterns and zones of moving path analysis [16–18]</td>
<td>- Social connection analysis (chatting, trade)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Coordinate analysis</td>
<td>- Merit: high feasibility</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Zone analysis</td>
<td>- Demeint: low accuracy rate</td>
</tr>
<tr>
<td></td>
<td>Human observation proofs (HOP) analysis</td>
<td>Detection method with keyboard and mouse input patterns analysis [19,20]</td>
<td>- User inputs observation</td>
<td>- Merit: high accuracy rate</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>- Windows event sequence analysis</td>
<td>- Demeint: low feasibility</td>
</tr>
</tbody>
</table>

Figure 4.2: Table 2: Recent Research on Bot Detection [13]

There is still room to improve on bot detection methods as none are perfect. That is why many games today are still plagued with bot users ruins the game for non-botting players. When developing a detection method it is crucial to have as little impact on the game experience as possible, not to take up too much processing power, and respect the player’s privacy. Due to these facts, the best place to detect bots is server side detection as it does not interfere with the player’s game experience or privacy in any way. Bot companies are constantly programming their product to avoid being detected. Detection methods that focus on behaviour/answers that are hard for a bot to mimic will last longer and not be countered by the next update for the bot.

A different variation of bot detection works better depending on the platform or type of game. This paper will be looking at WoW, a MMORPG. CAPTCHA analysis does not lend itself well to this type of game as the user will constantly have to be tested for it to be effective but this will interrupt game play and enjoyment.
Bots have evolved past traffic analysis as they often delay sending packets to avoid detection as appear more like a human. Moving path analysis is also fading away as bots today change up their recorded route from time to time and never mine a specific root too often. HOP analysis is a promising area of research but needs further investigation. Due to these facts, the best candidate out of all the methods examined for bot detection is behavioural analysis. The next thing to examine is the best machine learning algorithm so use.
Chapter 5

Machine Learning and Data Mining

In this section we look at the fundamentals of machine learning and the best data mining algorithms. This will give us some insight as to what would be the best algorithm for us to use.

Learning with regards to machines involves the machine changing its structure, program or data (based on its inputs or in response to external information in such a manner that its expected future performance improves. Machine learning programs are structured to solve tasks associated with artificial intelligence (AI) These changes they refer to might be either enhancements to already performing systems or ad initio synthesis of new systems. Most of the models used in machine learning are based of certain aspects of biological learning [30,31].

The primary goal of machine learning research is to develop general approaches to solve different learning problems. Theses learning problems could include:

- Optical character recognition: categorize images of handwritten characters by the letters represented
- Face detection: find faces in images (or indicate if a face is present)
- Spam filtering: identify email messages as spam or non-spam
Bot Detection Using Machine Learning Algorithms

- Topic spotting: categorize news articles (say) as to whether they are about politics, sports, entertainment, etc.

- Spoken language understanding: within the context of a limited domain, determine the meaning of something uttered by a speaker to the extent that it can be classified into one of a fixed set of categories

- Medical diagnosis: diagnose a patient as a sufferer or non-sufferer of some disease

- Customer segmentation: predict, for instance, which customers will respond to a particular promotion

- Fraud detection: identify credit card transactions (for instance) which may be fraudulent in nature • weather prediction: predict, for instance, whether or not it will rain tomorrow [32].

5.1 Major Types of Machine Learning

5.1.1 Supervised Learning

Supervised learning involves having already knowing the values of a set $f$, which a part of a larger set $E$. From this sample set $f$ it constructs a hypothesis, $h$, that agrees with the already know values of, $f$. This hypothesis, $h$, is then applied to the reminder of the unknown samples in $E$. The values of $E$ are not always correct but are a best guess based off of the knowledge we have gained from $f$. An example of supervised learning is curve-fitting: Suppose you are given points belonging to a two-dimensional function $f$. You want to construct a function $h$, where $h$ belongs to the set of second-degree functions that would incorporate all the points already given to you. You show there a two-dimensional parabolic surface above the $x_1, x_2$ planes that fits all the points. That parabolic surface is the hypothesis you have constructed from you already known variables [33,34].
5.1.2 Unsupervised learning

In unsupervised learning, we simply have a training set of vectors without function values for them. Unsupervised learning then partitions the training set into subsets in some appropriate way. This method of learning still revolves around learning a function, the value of the function is the name of the subset to which an input vector belongs. Unsupervised learning is typically used to cluster and label significant data [33,34].

5.2 General Problems Data Mining Algorithms

Solve

**Regression:** the goal of a regression based machine learning program is to estimate a real-valued variable $y \sigma R$ given a pattern $x$. An example of this is to estimate the value of stock in the upcoming days, the yield of a semiconductor fabrication plant given the current process, or the iron content of ore given mass spectroscopy measurements [33].

**Classification:** the goal of a classification based machine learning program is to answer the question: given a pattern $x$ drawn from a domain $x$, estimate which value an associated random variable $y \in \{1, \ldots, n\}$ will assume. This has much practical application. A simple example of this a bank wanting to know if a home owner might default on his loan. The bank already know information about the home owner such as income data, his credit history, etc. using this information as the pattern $x$ it can be estimated whether the home owner, $y$, will default on his loan or not default, not default [33].

**Clustering:** the goal of a classification based machine learning program is to group data sets into subsets. A typical approach for clustering would be to identify how different one instance of that data set is from another. It works out the distance bases on information about the instances. It groups the closest instances together and moves on till all instances belong to a cluster. An example of this is constructing the wildlife families. We take the different attributes about animals: Diet, gestation periods, number if toes, etc. Using this information we grope the different animals into their families [33].
5.3 Top data mining algorithms

Xindong Wu et al. listed the top 10 algorithms in data mining [30]. The following section looks at the algorithms that can be applied to detecting bots and discusses whether or not it’s the best solution to the problem. The problem of detecting bots can be broken down into a classification or a clustering problem. Classification in this case refers to training a program to be able to tell an input as either a human or a bot. Clustering on the other hand, can take all the data gathered in the experiment and cluster it into two groups, one would be the bots and the other the humans. When new data comes in it would test to see which cluster it belongs to.

5.3.1 C4.5 algorithm

The top ranked data mining algorithms was the C4.5 algorithm for building a decision tree. A decision tree classifies an instance by sorting them down the tree from the root (top of the tree) to some leaf node (bottom of the tree), which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for the attribute. In our example, the instances would be the data gathered from a WoW session. The attributes would be the faming factors that humans and bot seem to differ on.

The algorithm for the C4.5 decision tree is [35]:

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
• Conditions for stopping partitioning
  – All samples for a given node belong to the same class
  – There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  – There are no samples left

Information gain represents to what degree an attribute can be used to classify data. The attribute with the highest information gain is put as the first leaf of the tree, because it does the most work in classifying the data.

To work out the information gain we use the following formula:

\[
\text{Entropy} = -p(a) \times \log_2(p(a)) - p(b) \times \log_2(p(b))
\]

(5.1)

Where \( p(a) \) is the probability of \( a \) in the given set.

\[
\text{Information Gain} (T, X) = \text{Entropy}(T) - \text{Entropy}(T, X)
\]

(5.2)

This information is extremely useful not only for constructing our decision tree but for telling us which factor is contributing the least to the decision tree and we can see what impact it will have on the classification if it is taken out [36].

After the tree is constructed it is then pruned, a technique designed to reduces the size of the decision tree by taking away sections that do not contribute much to the classification process. In doing so pruning reduces overfitting.

The pruning algorithm in C4.5 is based on the estimate of the error rate associated with a set of \( N \) cases, \( E \) of which do not belong to the most frequent class. The algorithm calculates the upper limit of the binomial probability when \( E \) events have gone through \( N \) trails and the results recorded, using a default confidence value of 0.25.
Pruning is then carried out from the leaves to the root. The estimated error for a leaf tested with N cases and E errors is N times the error rate calculated above. The algorithm looks at the subtree and adds all the estimated errors of the branches together, call it H, and compares it to the estimated error if the subtree was a leaf, call it T. If it turns out that H is greater than T, the subtree will be pruned. The algorithm also checks if the estimated error if the subtree was replaced by one of its branches, if this appears to be beneficial then the subtree will be replaced [36].

5.3.2 The k-means algorithm

K-means clustering is an algorithm to classify or to group your objects based on attributes/features into k number of groups/clusters. The number of groups depends on the k where k is a positive integer and works as follows. A designated centroid or centre of the clusters is designated for each cluster. This can take the form of a random data point or the first data point in the sequence to be looked at. Treat each of these centroids as their own cluster. The algorithm then iterates between two steps:

Step 1: The distance between all the data points and the centre of all the different clusters is calculated. The default distance algorithm is the Euclidean distance formula. Group each data point to their closest cluster, with ties broken arbitrarily.

Step 2: After each data point has been clustered into their respective clusters, the algorithm recalculates the centroid of the cluster based on the newly assigned data point. If the data points come with a probability measure (weights), then the recalculated centroid is calculated using these weights (weighted mean).

The algorithm keeps iterating between these two steps until there is no change in the assignment of data points to clusters.

One of the main short falls of this approach is it will converge to a local optimum, so the cluster could depend on the location of the initial centroids. This can be countered to an extent by running the algorithms multiple times, each one with different starting centroids, or by doing limited local search about the converged solution [35-36].
5.3.3 Support vector machines (SVM)

For a binary classification problem a support vector machine finds the best function to distinguish between members of the two classes. It achieves this by finding the hyperplanes that separate the data. Of course there are many such hyperplanes that accomplish this task but support vector machines strive to find the one that creates the largest margin between the two classes. The margin is the amount of space between the two classes separated by the hyperplane.

The way it accomplishes this is by maximizing the following function with respect to $\vec{w}$ (w vector) and $b$:

$$L_p = \frac{1}{2} ||\vec{w}|| - \sum_{i=1}^{t} \alpha_i y_i (\vec{w} \cdot \vec{x}_i + b) + \sum_{i=1}^{t} \alpha_i$$

(5.3)

where $t$ is the number of training examples, and $\alpha_i, i = 1, ..., t$, are non-negative numbers such that the derivatives of $L_p$ with respect to $\alpha_i$ are zero. $\alpha_i$ are the Lagrange multipliers and $L_p$ is called the Lagrangian. In this equation, the vector $\vec{w}$ and constant $b$ define the hyperplane [36].

5.3.4 The EM algorithm

Finite mixture models are being increasingly used to model the distributions of a wide variety of random phenomena and to cluster data sets. Xindong Wu et al focused on mixture models, which can be used to cluster continuous data and to estimate the underlying density function. These models can be fitted by maximum likelihood via the EM (Expectation-Maximization) algorithm.

We will talk about its relevance to cluster analyses. The EM algorithm laid out by Xindong Wu et al. is as follows:
We let the p-dimensional vector \( y = (y_1, ..., y_p)^T \) contain the values of \( p \) variables measured on each of \( n \) (independent) entities to be clustered, and we let \( y_j \) denote the value of \( y \) corresponding to the \( j^{\text{th}} \) entity \( (j = 1, ..., n) \). With the mixture approach to clustering, \( y_1, ..., y_n \) are assumed to be an observed random sample from mixture of a finite number, say \( g \), of groups in some unknown proportions \( \pi_1, ..., \pi_g \). The mixture density of \( y_j \) is expressed as:

\[
f(y_i) = g_i = 1\pi_i f_i(y_j; \theta_i)(j = 1, ..., n)
\]

(5.4)

Where the mixing proportions \( \pi_1, ..., \pi_g \) sum to one and the group-conditional density \( f_i(y_j; \theta_i) \) is specified up to a vector \( \theta_i \) of unknown parameters \( (i = 1, ..., g) \). The vector of all the unknown parameters is given by:

\[

(\pi_1, ..., \pi_{g-1}, \theta^{T_1}, ..., \theta^{T_g})^T
\]

(5.5)

where the superscript “T” denotes vector transpose. Using an estimate of \( \pi \), this approach gives a probabilistic clustering of the data into \( g \) clusters in terms of estimates of the posterior probabilities of component membership,

\[

\tau_i(y_j) = \pi_i f_i(y_j; \theta_i) f(y_j)
\]

(5.6)

where \( \tau_i(y_j) \) is the posterior probability that \( y_j \) (really the entity with observation \( y_j \)) belongs to the \( i^{\text{th}} \) component of the mixture \( (i = 1, ..., g; j = 1, ..., n) \). The parameter vector can be estimated by maximum likelihood. The maximum likelihood estimate (MLE) of \( \pi \), \( \hat{\pi} \), is given by an appropriate root of the likelihood equation:

\[

\frac{\partial \log L()}{\partial} = 0
\]

(5.7)

where \( \log \)

\[

L() = n_j = \log f(y_j)
\]

(5.8)

is the log likelihood function for \( \pi \). Solutions of (6) corresponding to local maximizers can be obtained via the expectation–maximization (EM) algorithm [17]. For the modeling of continuous data, the component-conditional densities are usually taken to belong to the same parametric family, for example, the normal. In this case,
\[ f_i(y_j; \theta_i) = \phi(y_j; \mu_i, i) \]  

where \( \phi(y_j; \mu_i, i) \) denotes the \( p \)-dimensional multivariate normal distribution with mean vector \( \mu \) and covariance matrix”.

### 5.3.5 AdaBoost

Ada boost is a type of ensemble learning algorithm which means it creates multiple learners to solve a problem. It is a very popular algorithm with very accurate prediction and can be programmed in just 10 lines of code.

Let \( X \) be the instance space and \( Y \) the set of class labels (in our case \{bot, human\}). The algorithm assigns equal weights to all the training examples. The algorithm creates a base /learning function \( f : X \rightarrow Y \). This base learner is tested with all the training examples. The weights of the incorrectly classified examples will be increases. The algorithm creates another bases learning function this time taking into account the updated weights. This process is iterated over a certain number of rounds. The final model is derived by weighted majority voting of all the base learning functions, the weights of the learners are determined during the training process [36].

### 5.3.6 kNN

K-Nearest neighbour or kNN is typical used as a classification algorithm. The \( k \) term represents a number that decides how many neighbours influence the classification and is an odd number to eliminate the possibility of a tie condition. How it works is when \( k \) is 1 it takes a point \( x \) and finds the closest point to \( x \), let’s call it \( y \). The label of \( y \) is already known. It labels \( x \) as the same as \( y \).
When $k$ is greater than 1 we try to find the $k$ nearest neighbours and do a majority voting. Sometimes not all points’ votes carry the same weighting, this is called weighted kNN. The weighting of a point’s vote is calculated by taking the inverse distance weighting to that point. This lets point closer to the data point carry more weight than those further away. Weights can also be assigned based on the reliability of the data letting more reliable data carry a higher weight.

KNN classifiers are lazy learners. This means the models do not require much computational power to build, however classifying data points requires a lot of computational power. Every time the algorithm needs to classify a data point is will have to compute the k-nearest neighbours for that data point.

The algorithm used for majority voting is:

$$\text{Majority Voting } y' = \arg\max \sum_{(x_i, y_i) \in D_z} I(v = y_i) \tag{5.10}$$

where $v$ is a class label, $y_i$ is the class label for the $i$th nearest neighbors, and $I(\cdot)$ is an indicator function that returns the value 1 if its argument is true and 0 otherwise.

The overall algorithm can be summed up as follows:
There are various factors that affect the performance of KNN. One such factor is the $k$ value. If the $k$ value is too small, then the result can be easily influenced by noise points. If the $k$ value is too large, then the majority voting can be influenced by too many point form other classes. The method for determining distance depends on the parameters of the experiments. The Euclidean distance can be affected by the high dimensionality of the data. Euclidean distance becomes less discriminating as the number of attributes increases. Sometimes it is necessary to scale the attributes of the data, as one attributes might span a much greater distance than another. This causes one attributes to carry more weight than the others [34-36].
5.3.7 Naïve Bayes

A naïve Bayes is a supervised learning algorithm that works well with large data sets. A Naïve Bayes classifier works with the Bayes theorem, a probability theorem which can be seen is defined as:

\[
P(H|E) = \frac{P(H) \times P(E|H)}{P(E)}
\]  

(5.11)

In this equation, the posterior probability \( P(H|E) \) is the probability of a label (bot or human) given the data \( E \) (the data from a WoW session). \( P(E|H) \) is the probability of that data given a the label \( H \). \( P(H) \) is the probability of that label. \( P(E) \) is the probability of the data. In our case \( E \) is made up of 5 different attributes. To work out \( P(E|H) \) we used the formula \( P(E|H) = P(E_1|H) \times P(E_2|H) \times \ldots \times P(E_5|H) \). This formula only works if the attributes are independent of each other, which is not the case with our attributes [37].

5.3.8 CART

CART stands for Classification and Regression Trees. It is a binary recursive partitioning procedure capable of processing continuous and nominal attributes both as targets and predictors. Data can be used in its raw form. The tree is grown to a maximal size as no stopping rule is implemented. Once the tree is fully grown then certain sections are removed according to a cost-complexity pruning formula. The split found to contribute the least to the classification according to the training set is then removed.

The CART process creates more than one tree. A sequence of nested pruned trees all of which are candidate optimal trees. The optimal tree is decided by their predictive performance in the pruning sequence. Tree performance is always measured via cross validation and selection takes place only after test-data-based evaluation. If there is one test data to perform the cross validation, the program will be unsure of the best tree in the sequence to select [36].
5.3.9 Artificial neural network (ANN)

ANN is a network inspired by biological neural networks (the brain). By looking at how animals make decision we mimic that process to create an artificial neural network that can be used to approximate functions that can depend on a large number of inputs that are generally unknown.

The basic approach to neural networks is the inputs are all multiplied by weights (representing the strength of the input), and then computed by a mathematical function which determines the activation of the neuron. By adjusting the weights of the inputs, how much each factor contributes to the decision, we can obtain the output we want for specific inputs. Sometimes there are hundreds of thousands of neurons so it would be impractical to adjust all the weights by hand. The ANN adjusts the weights but a process called back propagation. With back propagation data sets of known values are given to the ANN and the weights are adjusted in such a way as to give the correct output for the data sets [38].
Chapter 6
Methodology

In this paper we will be looking at a bot detection approach for WoW. This experiment looks at classifying a WoW character based on human behaviour that a piece of software would struggle to mimic. The behaviour this paper covers is efficiency. Farming efficiently can be broken down into 5 different aspects.

- Percent of Items Gathered: Most human players will get distracted and miss/not see an item drop, whereas bots will almost always detect and head towards the item drop.

- Distance between the player and item drop when farming: The distance between the WoW character and the item drop when they start harvesting is also a factor. With human players the exact distance between the WoW character and the item drop will vary, while the bots will show a pattern.

- Enemies engaged: Bots typically do not go after an item if there are too many enemies in the vicinity, whereas a human player might consider it.

- Time between item collections. Bots typically fly according to a predefined path, and because of this, a pattern emerges regarding the time it takes to collect an item.

- What drops are gathered: Some drops are hidden/obscured by other objects in the game. Bots will still go to these items while human plays might not see them.
This paper proposes, using these different attributes, ways to classify an in-game character as bot-controlled or human-controlled.

6.1 Research Questions

- Can bots be identified to a sufficient level by using efficiency factors related to gold farming?
- Can bots be identified to a sufficient level by using fewer efficiency factors related to gold farming?
- How well does the proposed method deal with bot profile outliers, can it identify a bot that is setup in an illogical way and detrimental to the efficient farming of items?

6.2 Conditions for the experiment

6.2.1 The Location

WoW’s map is split according to a player’s level. An area in WoW can have different spawn rates for drops and enemies and this can have an effect on the data we wish to collect. If this bot detection method was to be implemented in the game each area would have to have its own classifier.

Traditionally the higher the areas level the better the items that can be farmed so normally have a higher chance of containing a bot. Players that use farming bot will most likely have a character at max level for the bot to control. Due to this facts, the experiment will take place in a high-level area (76-78) called Sholazar basin.

Although this is a high level area it does not mean that the players are high level players themselves. Many players of different skill level play WoW. Some players have levelled up multiple characters and have spent many hours playing the game. Most players that get to Sholazar basin have a high skill level as it is a level 76-78 area. Some players play on servers that multiply typical experience rates by a factor of up to 20, some could just be playing on a friend’s account, or have not played in a long time. The information gathered from Sholazar basin is from a wide variety of silk level and not just from advanced players.
6.2.2 The WoW server

The game environment was hosted privately to ensure a secure and isolated test environment. This is facilitated by a server repack titled “Jeutie’s Blizz-like repack version 3.3.5a”. This lets a private instance of World of Warcraft (WoW), The Wrath of the Lich King version 3.3.5a be hosted for research purposes only. The bots and the human players will play with the same WoW character to make sure the tests are reliable. When the participants and the bots are playing the game they will be alone in the word. This is obviously different from playing the real game, as on occasion, other players will mine the ore before you can. As this is the same for the bots and the human participants, it should not be a discerning factor on the outcome of classifying the controller of the WoW character.

Enemies found in Sholazar basin

- The mighty dreadsaber Shango.
- The black-furred lion Pitch.
- The towering green devilsaur King Krush.
- The poisonous emperor cobra Venomtip.
- The incredibly elusive and beautiful mate of Harkoa herself, the spirit beast Logue’nahak.
- The fiercely protective protodragon Broodmother Slivina.

6.2.3 Player Data Collectoin

The player information was collected from Blizzard. Blizzard provided two data bases, one with player information in Sholazar basin and the other data base was about the state of the area, enemies and drops. Using the two data bases in conjunction we were able to extract the necessary information.
This methodology targets players that are farming exclusively. Five minutes was the minimum amount of time that meaningful information could be gathered. Players that were not in Sholazar basin for five minutes were excluded from the list. Players were classified as farming if any specific item in their bag increased five in the five minute period. The minimum amount that a bot managed to increase their selected item by was eight given the same time frame. 97 player profiles met the required condition and were classified as farming.

6.2.4 The bot programs

The experiment will involve two different bot programs, Lazy bot and Honorbuddy. Lazy bot is a free to download bot program so it attracts lots of players that are not willing to pay money for a bot. Honorbuddy is rated one of the top WoW bots and has over 260,000 registered users. Both bots will be run with using different behaviour scripts. The bots will do 108 test cases. Each bot will have three different routes around Sholazar Basin. Each bot will be tested with two different “Do not approach target with certain number of enemies around” (DNA) number. Each bot will be tested with three different combat approaches, casting different spells, when to use positions, when to run away, etc. Each route will be tested three times starting at a different location throughout Sholazar basin. The bots main function will be to gather Saronite ore.

6.3 Bot data collection

The bot information was collected on a private server.

The WoW server:

This is facilitated by a server repack titled “Jeutie’s Blizzlike repack version 3.3.5a”. This lets a private instance of World of Warcraft (WoW), The Wrath of the Lich King version 3.3.5a be hosted for research purposes only. The bots will play with the various WoW character types just like you would find in the real server. They were also equipped with different items such that the gear score would range between 2850 an 3200.
When the bots are playing the game they will not be alone in the world as their will be other bots also farming. From the player information it was calculated that on average there are 1.2 other WoW players farming the same item. To emulate this factor 3 bot would farm at the same time. Two bots would farm for the whole 5 minutes and one would farm for one minute. The information of the bots farming for five minutes would be saved.

Various pieces of information are required to carry out this experiment. The program needs to collect the location and number of the different enemies surrounding item drops, the location of the player, the location and number of item drops surrounding the player, which item drops the player gathered, the time the items were gathered, and when the player is in combat.

Lazy Bot already collects this information from WoW as it needs this information as well to perform its mining function. The information was easier to collect from the Lazy bot’s database than it was from WoW’s database. Honorbuddy will have Lazy bot open in the background but not running to make recording information easier.

A program will be running while the test is carried out. This program will read the Lazy bot database and save the aforementioned information every second. The program recognizes the item as collected if it is removed from the Lazy bot database. As soon as an item is collected the distance between the WoW characters and the items location is saved. The time at which the item is removed from the database is also saved. If the player comes within a certain range of an item drop the program will save it as a possible collection. If the WoW characters goes and mines that item drop than it registers it as an achieved collection. If the player enters combat, a combat flag will be saved for as long as the player is in combat.
6.3.1 Data pre-processing

Each test case must have 5 different attributes: percent gathered, enemies engaged, distance from drop, time taken between drops, and location of drops. These attributes can be either numerical or nominal and this will determine the format of the test. If the value is numeric, $A$ then the format is normal of the type $\{A < h, A > h\}$ where the threshold $h$ value calculated to maximise the classification between the bots and humans. For a nominal test it is of the form if value $A \in Y$, where $Y$ is a set containing general bot attributes, then it gets classified as a bot, otherwise it get classified as a human.

Data pre-processing is the process of transforming our raw data into the attributes that can be used to test if it is a human or a bot.

Percent gathered is calculated by possible collection/achieved collection. Possible collection refers to all the drop the player come within a certain range of. One problem is that bots are normally programmed not to approach an item drop if there are too many enemies around. This factor could disrupt our classification. Any drops that have a potential threat of entering combat with enemies will be excluded from this data set.

To work out the number of enemies engaged the program looks for when the WoW character and an enemy are with in close proximity of each other and the WoW character is in combat. When this happens the program marks it as an encounter with so many different enemies and saves that number. The encounter is over when the player and all enemies are a certain distance apart. The enemies engaged value will be the sum of all the enemy encounters.

For distance between the player and item drop when farming the program finds out what the distance between the WoW character and the desired item drop is, the drop he is closest to. This is accomplished by applying the distance formula to the WoW characters coordinates and the last known location of the item that has just disappeared from the data base, the drop that has been farmed.
Time taken between drops is saved to the data base directly and is calculated as the time between when the WoW character is registered as farming. The standard deviation proved to more effective than the mean when testing the classifier.

The last piece of information need to be pre-processed is the location of drops. All the drop locations that have been successfully mined during a session will be saved into an array. Each location get a score, for every bot that has farmed that location it gets a plus one, for every human a minus one. For every location that a subject farms that location scores gets added up. The possibility arises that one location can gain a very high score and can dominated all the other location that the subject farms. To avoid this problem we limited the location score to maximum of 5 and minimum of -5.

### 6.3.2 Classifier architecture and training

There are many different machines learning classifier to choose from, after testing out the data set on various classifiers it was found decision tree delivered the best results.

A C.4.5 decision tree algorithm was to construct the tree. This algorithm was voted the most efficient classifying algorithm at the ICDM in 2006 (30). I will be using the Weka program to build the tree and analyse the results.

The attributes that the test cases will be split on are: percent gathered, enemies engaged, distance from drop, time taken between drops, and location of drops.

The initial classifier will be tested in three different ways.

- The first test will involve training the classifier with 43 bot and 38 human cases and tested with 65 bot and 59 human cases.
• The 2nd test will involve training the classifier with 65 bot and 59 human cases and tested with 43 bot and 38 human cases. This test will be carried out four times, each one with a randomized training set and average of the results will be taken. This is to ensure that cases misclassified in the first test are not just put in the training sample for the second test.

• The 3rd test will repeat test one and two but tested with 30 new bot profiles a lot of these cases are outliers, bots that are set up in an illogical way. This is to test how well the classifier deals with bot outliers.

All test mentioned above were be carried out again this time the farming factor deemed lest important to the classifier will be taken out.

Based on the result of the above test another test was carried out. This test involved adding an extra attribute to the tree call way points visited.

The waypoints visited attribute is used to determine how often a WoW character was in ranged of certain location. Bots normally follow a path constructed by a set of way points. They do not always have to follow a straight line or cross over the way point directly, this makes it harder to determine if a WoW character is a bot bases on its movement.

The program marks each location the WoW character visits as a centre and constructs a circle of a certain radius around it; this is registered as a way point. A new way point will only be constructed when the player is a certain radius away from the centre of the old way point and construct a new circle as a way point.

The data base already contained the location of the WoW character. The distance function was applied to WoW character location and the next location found to be certain radius away the location was saved. If two way points overlay the program saves it as a hit, this indicates the WoW character has returned to that way waypoint.
A human player could possibly just be passing the same point twice, while a bot is more likely to pass a continuous stream of way point at a time. If the WoW character passes through a way point previously entered, one point will be added to their waypoint visited count. If the WoW character passes through a way point previously entered and is also coming from a way point previously entered two points will be added to their waypoint visited count.

The WoW characters position has already been stored into the data base. It will take up some processing power to prepare the data to be used. We will not have to gather extra data from the WoW server to incorporate this extra attribute.

The extra attributes test will repeat the condition for test one and two and then repeat them with the bot outlier cases added.
Chapter 7

Results

The accuracy of a bot detection method is very important. Once an account is found to be using a bot harsh action is taken against that account such as banning or freezing the account for a time period. To have players that have spent money and time developing their character detected as bot and have their account deleted can be extremely detrimental to the games success. It is because of these reasons that a bot detection method must have a very low false positive rate.

Weka outputs a detailed accuracy report which includes: True positive rate, False positive rate, Precision, and F-measure and a confusion matrix.

With this information it will be easy to judge how efficient the algorithm is.

True Positive (TP) Rate: The percentage that the human cases were classified as humans.
TP rate is calculated by TP/(TP+FN).

False Positive (FP) Rate: The percentage that the bot cases were classified as human.
FP rate is calculated by FP/(FP+TN).

Precision: The percentage of cases that were classified as human those are actually human.
Precision is calculated using the following formula = TP/(TP+FP).

Accuracy: The percentage of the test cases that were classified correctly.
Accuracy is calculated by (TP+TN)/(TP+TN+FP+FN).

F-measure: The harmonic mean between precision and recall.
F-measure is calculated in the following way $F = \frac{2 \times \text{Precision} \times \text{recall}}{\text{Precision} + \text{recall}}$.

### 7.1 Calculating the information gain for each attribute

The attributes that the test cases will be split on are: percent gathered, enemies engaged, distance from drop, time taken between drops, and location of drops. As stated earlier information gain is calculated using the follow two formulae:

$$\text{Entropy} = -p(a) \times \log_2(p(a)) - p(b) \times \log_2(p(b)) \quad (7.1)$$

Where $p(a)$ is the probability of $a$ in the given set.

$$\text{Information Gain } (T, X) = \text{Entropy}(T) - \text{Entropy}(T, X) \quad (7.2)$$

The results of calculating the information gain are as follows:
- Ranked attributes: Distance from drop, Location of drops, Enemies engaged, Time taken between drops, Location of drops

### 7.2 First Test

The first test (1) involved training the classifier with 43 bot and 38 human cases and tested with 65 bots and 59 human cases. This was taken as a base test to see how changing factors in the next tests will influence the results.

The first thing to consider is the confusion matrix:

```plaintext
=== Confusion Matrix ===
```
Where ‘a’ represents humans and ‘b’ represents bots.

The results of the first test decision tree were that it classified 45 cases as humans and 14 as bots from the 59 human test cases. It classified 19 cases as humans and 46 as bots from the 65 bot test cases. This gives us the following information.

TP Rate = $\frac{45}{59} = 76.27\%$

FP Rate = $\frac{19}{65}=29.23\%$

Precision = $\frac{45}{64} = 70.31\%$

Accuracy = $\frac{91}{124} = 73.39\%$

F-measure = $\frac{30}{41} = 73.17\%$

### 7.3 Second test

The second test (2) involved training the classifier with 65 bot and 59 human cases and tested with 43 bot and 38 human cases. This test will be carried out four times, each one with a randomized training set and average of the results will be taken. This is to ensure that cases misclassified in the first test are not just put in the training sample for the second. This test is to see how greatly additional training cases will affect the classification ability of the tree.

<table>
<thead>
<tr>
<th></th>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>32</td>
<td>6</td>
<td>a</td>
</tr>
<tr>
<td>7</td>
<td>36</td>
<td>b</td>
</tr>
</tbody>
</table>
Bot Detection Using Machine Learning Algorithms

Where ‘a’ represents humans and ‘b’ represents bots.

TP Rate = 16/19 = 84.21%
FP Rate = 7/43 = 16.28%
Precision = 32/39 = 82.05%
Accuracy = 68/81 = 83.95%
F-measure = 64/77 = 83.12%

7.4 Missing attribute test

This test eliminates the attribute with the least information gain from the tree and repeats the past tests. The attribute with the least information gain was Location of drops. This test case will construct and test a tree with the following attributes only. Percent gathered, enemies engaged, distance from drop, and time taken between drops. This test will give us an idea if we can successfully classify a WoW character with less information and less processing power.

The Missing attribute test one (3.1) involved training the classifier with 43 bot and 38 human cases and tested with 65 bots and 59 human cases.

=== Confusion Matrix ===

\[
\begin{bmatrix}
a & b \\
42 & 19 \\
21 & 48 \\
a & b
\end{bmatrix}
\]

Where column ‘a’ represents humans and column ‘b’ represents bots.

TP Rate = 42/61 = 68.85%
FP Rate = 7/23 = 30.43%
Precision = 2/3 = 66.67%
Accuracy = 9/13 = 69.23%
F-measure = 21/31 = 67.74%
The Missing attribute test two (3.2) involved training the classifier with 65 bot and 59 human cases and tested with 43 bot and 38 human cases. As the same in test two, this test will be carried out four times, each one with a randomized training set and an average of the results will be taken.

\[
\begin{bmatrix}
a & b \\
29 & 9 \\
12 & 31 \\
\end{bmatrix}
\]

Where 'a' represents humans and 'b' represents bots.

TP Rate = \( \frac{29}{9} = 76.32\% \)
FP Rate = \( \frac{12}{43} = 27.91\% \)
Precision = \( \frac{29}{41} = 70.73\% \)
Accuracy = \( \frac{20}{27} = 74.07\% \)
F-measure = \( \frac{58}{79} = 73.42\% \)

7.5 Bot profile outliers

Bot profile outliers test one (4.1) involved training the classifier with 43 bot and 38 human cases and tested with 65 normal bots, 30 bot outliers, and 59 human cases.

\[
\begin{bmatrix}
a & b \\
45 & 14 \\
37 & 58 \\
\end{bmatrix}
\]

Where 'a' represents humans and 'b' represents bots.

TP Rate = \( \frac{45}{14} = 76.27\% \)
FP Rate = \( \frac{37}{95} = 38.95\% \)
Precision = \( \frac{45}{82} = 54.88\% \)
Bot Detection Using Machine Learning Algorithms

Accuracy = $103/154 = 66.88\%$
F-measure = $30/47 = 63.83\%$

**Bot profile outliers test (4.2)** two involved training the classifier with 65 bot and 59 human cases and tested with 43 bot and 38 human cases. This test will be carried out four times, each one with a randomized training set and average of the results will be taken. This is to ensure that cases misclassified in the first test are not just put in the training sample for the second.

```plaintext
=== Confusion Matrix ===

\[
\begin{bmatrix}
a & b \\
32 & 6 & a \\
22 & 51 & b \\
\end{bmatrix}
\]
```

Where 'a' represents humans and 'b' represents bots.

TP Rate = $16/19 = 84.21\%$
FP Rate = $22/73 = 30.14\%$
Precision = $16/27 = 59.26\%$
Accuracy = $83/111 = 74.77\%$
F-measure = $16/23 = 69.57\%$

7.6 Extra attribute test

Due to the fact that the best F-measure we could obtain was 83.12

**The Extra attribute test one (5.1)** involved training the classifier with 43 bot and 38 human cases and tested with 65 bots and 59 human cases.

```plaintext
=== Confusion Matrix ===

\[
\begin{bmatrix}
a & b \\
48 & 11 & a \\
15 & 50 & b \\
\end{bmatrix}
\]
```

Where 'a' represents humans and 'b' represents bots.
Bot Detection Using Machine Learning Algorithms

TP Rate = 48/59 = 81.36%
FP Rate = 3/13 = 23.08%
Precision = 16/21 = 76.19%
Accuracy = 49/62 = 79.03%
F-measure = 48/61 = 78.69%

The Extra attribute test two (5.2) involved training the classifier with 65 bot and 59 human cases and tested with 43 bot and 38 human cases. As the same in test two, this test will be carried out four times, each one with a randomized training set and an average of the results will be taken.

\[
\begin{bmatrix}
a & b \\
33 & 5 \\
5 & 38
\end{bmatrix}
\]

Where ‘a’ represents humans and ‘b’ represents bots.

TP Rate = 33/38 = 86.84%
FP Rate = 5/43 = 11.63%
Precision = 33/38 = 86.84%
Accuracy = 71/81 = 87.65%
F-measure = 33/38 = 86.84%

The Extra attribute test three(5.3) involved training the classifier with 43 bot and 38 human cases and tested with 65 normal bots, 30 bot outliers, and 59 human cases.

\[
\begin{bmatrix}
a & b \\
48 & 11 \\
27 & 68
\end{bmatrix}
\]

Where ‘a’ represents humans and ‘b’ represents bots.

TP Rate = 48/59 = 81.36%
FP Rate = 27/95 = 28.42%
Bot Detection Using Machine Learning Algorithms

Precision = 16/25 = 64%
Accuracy = 58/77 = 75.32%
F-measure = 48/67 = 71.64%

The Extra attribute test four(5.4) involved training the classifier with 65 bot and 59 human cases and tested with 43 normal bot, 30 bot outliers, and 38 human cases. This test will be carried out four times, each one with a randomized training set and average of the results will be taken. This is to ensure that cases misclassified in the first test are not just put in the training sample for the second.

=== Confusion Matrix ===

\[
\begin{bmatrix}
  a & b \\
  33 & 5 \\
  14 & 59
\end{bmatrix}
\]

Where ‘a’ represents humans and ‘b’ represents bots.

TP Rate = 33/38 = 86.84%
FP Rate = 14/73 = 19.18%
Precision = 33/47 = 70.21%
Accuracy = 92/111 = 82.88%
F-measure = 66/85 = 77.65%

Table 3:
Results with normal bot cases

<table>
<thead>
<tr>
<th></th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3.1</th>
<th>Test 3.2</th>
<th>Test 5.1</th>
<th>Test 5.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP Rate</td>
<td>76.27%</td>
<td>84.21%</td>
<td>68.85%</td>
<td>76.32%</td>
<td>81.36%</td>
<td>86.84%</td>
</tr>
<tr>
<td>FP Rate</td>
<td>29.23%</td>
<td>16.28%</td>
<td>30.43%</td>
<td>27.91%</td>
<td>23.08%</td>
<td>11.63%</td>
</tr>
<tr>
<td>Precision</td>
<td>70.31%</td>
<td>82.05%</td>
<td>66.67%</td>
<td>70.73%</td>
<td>76.19%</td>
<td>86.84%</td>
</tr>
<tr>
<td>Accuracy</td>
<td>73.39%</td>
<td>83.95%</td>
<td>69.23%</td>
<td>74.07%</td>
<td>79.03%</td>
<td>87.65%</td>
</tr>
<tr>
<td>F-measure</td>
<td>73.17%</td>
<td>83.12%</td>
<td>67.74%</td>
<td>73.42%</td>
<td>78.69%</td>
<td>86.84%</td>
</tr>
</tbody>
</table>

Chapter 7 50 Ian D. Stevens
Table 4:
Results with normal bot bot outlier cases

<table>
<thead>
<tr>
<th></th>
<th>Test 4.1</th>
<th>Test 4.2</th>
<th>Test 5.3</th>
<th>Test 5.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP Rate</td>
<td>76.27%</td>
<td>84.21%</td>
<td>81.36%</td>
<td>86.84%</td>
</tr>
<tr>
<td>FP Rate</td>
<td>38.95%</td>
<td>30.14%</td>
<td>28.42%</td>
<td>19.18%</td>
</tr>
<tr>
<td>Precision</td>
<td>54.88%</td>
<td>59.26%</td>
<td>64%</td>
<td>70.21%</td>
</tr>
<tr>
<td>Accuracy</td>
<td>66.88%</td>
<td>72.77%</td>
<td>75.32%</td>
<td>82.88%</td>
</tr>
<tr>
<td>F-measure</td>
<td>63.83%</td>
<td>69.57%</td>
<td>71.64%</td>
<td>77.65%</td>
</tr>
</tbody>
</table>
Chapter 8

Discussion

Analysing the difference between tests one and two various measures displayed a notable increase as seen in table 5, as was hypothesised. We saw that the identification of bots significantly increased as well as the misclassification of human players decrease, from test one to test two, with the addition of 23 extra bot cases and 21 more human cases during the testing phase. Such a larger increase in performance indicates the fact that our test data was not large enough to train the classifier to its most optimum state. Unfortunately, 97 player’s cases were all we could get from Blizzard.

An increase in the size of the test data will almost always increase the results. Such is the nature of machine learning algorithms but with a larger data set, such a significant increase would not occur. One could expect significantly more impressive results should we increase the training sets further, perhaps performance even comparable to the top bot detection methods implemented in games today.

When comparing the tests, involving taking out the attribute “Location of drops”, the results diminished considerably, as seen in table 5, even though it had the lowest information gain. This goes against our initial hypothesis that the classifier might be able to classify WoW characters to an acceptable standard.
The classifier did poorly when asked to deal with bot cases that were programmed irrationally. The main static to focus on in these cases is the false positive rate, the rest of the data get skewed because of the much larger amount of bot cases than human cases. As seen in table 6 the difference of the false positive rate, between a test with the outlier bot cases and normal bot cases was over 9%. For the bot outlier test part one it classified less than half of the outliers as bots correctly.

Going back and looking at the log data of the bot outliers, the misclassification of bots can be attributed to two factors. Some bots programmed route is short and does not cover much area. In some cases the bots only possible items to farm is one surrounded by enemies. This in turn causes “Enemies engaged” and “Time between item collections” to be higher than normal and much closer to that of a human than a bot. The other factor is the poor combat approach some of the bots have; this led to the death of some bots and had the same effect on “Enemies engaged” and “Time between item collections” as the poor rout. As you can see in table 6, the classifiers ability to deals with these bot cases significantly improve with the addition of the attribute “waypoints visited”. These bot have a high score for repeated way points as they keep repeating their rout or die and have to return to their bodies.

When looking at what bots were correctly classified it was discovered that the majority of them came from the LazyBot program. In the extra attribute experiment with the larger training size, it correctly classified all the LazyBot cases. This is to be expected as LazyBot is a free to download program and a person would have to pay for HonorBuddy. The classifier did increase its proficiency at classifying HonorBuddy cases when the training size was increased, so the classifier would need more test cases to increase its result for more human-like bots.
## Table 5:
### Difference in results with normal bot cases

<table>
<thead>
<tr>
<th></th>
<th>Test 1</th>
<th>Test 2</th>
<th>Test 3.1</th>
<th>Test 3.2</th>
<th>Test 5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP Rate</td>
<td>7.94%</td>
<td>-7.42%</td>
<td>-7.89%</td>
<td>5.09%</td>
<td>2.63%</td>
</tr>
<tr>
<td>FP Rate</td>
<td>-12.95%</td>
<td>1.2%</td>
<td>11.63%</td>
<td>-6.15%</td>
<td>-4.65%</td>
</tr>
<tr>
<td>Precision</td>
<td>11.74%</td>
<td>-3.64%</td>
<td>-11.32%</td>
<td>5.88%</td>
<td>4.79%</td>
</tr>
<tr>
<td>Accuracy</td>
<td>10.56%</td>
<td>-4.16%</td>
<td>-9.88%</td>
<td>5.64%</td>
<td>3.7%</td>
</tr>
<tr>
<td>F-measure</td>
<td>9.95%</td>
<td>-5.43%</td>
<td>-9.7%</td>
<td>5.52%</td>
<td>3.72%</td>
</tr>
</tbody>
</table>

## Table 6:
### Difference in results with bot outliers

<table>
<thead>
<tr>
<th></th>
<th>Test 4.1</th>
<th>Test 4.2</th>
<th>Test 5.3</th>
<th>Test 5.4</th>
</tr>
</thead>
<tbody>
<tr>
<td>TP Rate</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
<td>0%</td>
</tr>
<tr>
<td>FP Rate</td>
<td>9.72%</td>
<td>13.86%</td>
<td>5.34%</td>
<td>7.55%</td>
</tr>
<tr>
<td>Precision</td>
<td>-15.43%</td>
<td>-22.79%</td>
<td>-12.19%</td>
<td>-16.63%</td>
</tr>
<tr>
<td>Accuracy</td>
<td>-6.51%</td>
<td>-11.18%</td>
<td>-3.71%</td>
<td>-4.77%</td>
</tr>
<tr>
<td>F-measure</td>
<td>-9.34%</td>
<td>-13.55%</td>
<td>-7.05%</td>
<td>-9.19%</td>
</tr>
</tbody>
</table>
Chapter 9

Future Work

As discussed in the previous section, testing this method with a larger data set would be the next step. If results do not improve with a larger data set adding extra attributes to the classifier could a solution. As seen in the tests, taking away and then adding an attribute changed the results significantly.

This method was designed with MMO in general in mind. The attributes tested in this experiment are present in all common MMO and the data need to generate these attributes are easily gathered. Bots are a common problem in all MMOs and it would be interesting to see if the same bot detection structure could be applied to these other MMOs and compare the results.

It would be necessary to see if bot companies could program their product in a way to carry out ongoing undetected with the classifier installed. We believe our classifier has chosen attributes of game play that would be hard for a bot to mimic and currently do not but there are constant advances in AI. If it was discovered what aspects of game play our classifier looks at, the bot companies might be able to programme their product to appear human to our classifier.
The attributes used in the construction of the decision tree are tailored specifically at bots used to farm items. As stated previously, bots can be used to do a large variety of activities. It would be interesting to see how, if factors of other gameplay, that would be hard for a bot to mimic, could be used to construct a tree. WoW characters would be split into categories based on their in-game aim, after they have been split they could be tested on a classifier specifically for their in-game aim, not unlike the method presented by Yeounh Chung et al.
Chapter 10

Conclusion

In this paper, we proposed a bot detection method. The method revolved around looking at various aspects of farming that a bot would find hard to mimic. Based on these aspects, a decision tree was constructed to classify a WoW character as either a bot or a human. Various experiments were conducted on the classifier mainly involving changing the training size or the addition or subtraction of an attribute to see the extent they would change the results.

We set out to answer the following research questions:

Can bots be identified to a sufficient level, in comparison to other methods, by using efficiency factors related to gold farming?

The best results obtained throughout the experiment were an accuracy of 87.65% and a F-measure of 86.84%. Other bot detection methods have been able to obtain much higher results, upwards of 92%. Unfortunately, in this experiment we were only able to obtain a data set of 97 human cases and 108 bot cases. It was shown that this probably an insufficient data set to train and test the classifier. The test will need to be carried out again with a larger data set as the method does show promise. As the results stand at the moment, the proposed method comes close to that of top bot detection methods but is still inferior.
Can bots be identified to a sufficient level by using fewer efficiency factors related to gold farming?

When the factors with the lowest information gain (Location of drops) was removed from the classifier the results diminished by a considerable amount, 5% in test one and 7% in test two. Bot could not be sufficiently classified with fewer efficiency factors. This will still probably be the case even if the data size for training the algorithm increased.

How well does it classify bot profile outliers?

The initial test classified the majority of these bot cases as humans. Adding an extra attribute increased the efficiency of the classifier by 7%, in the test with a greater training size. To correctly classify these bot one would have to observe them and add more attributes that distinguish them from human players. As it stands the best FP rate we could achieve was 19.18% which is less than ideal.
Chapter 11

References


Bot Detection Using Machine Learning Algorithms


[17] Lu, Yuanchao, et al. ”MMOPRG bot detection based on traffic analysis.” International Journal of Electronics and Information Engineering 2.1


