VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. Road, V.V.Puram, Bengaluru-560 004

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Dissertation on

"PREDICTIVE ANALYSIS OF A REAL TIME STRATEGY GAME- Dota 2"

Submitted By

USN	Name
1BI13CS052	**Dwarakanandan B M**
1BI13CS050	**Divyendu Dutta**
1BI13CS036	**Bhargav Chamala**
1BI13CS016	**Anirudh M Bhat**

For the academic year 2016-17

Under the Guidance of

Dr.Suneetha K R

Associate Professor

Department of Computer Science & Engineering

Bangalore Institute of Technology

Bengaluru-560004

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY

Bengaluru -560 004

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Certificate

This is to certify that the Project work entitled **"Predictive analysis of a real-time strategy game-Dota2"** has been successfully completed by

USN	Name
1BI13CS052	**Dwarakanandan B M**
1BI13CS050	**Divyendu Dutta**
1BI13CS036	**Bhargav Chamala**
1BI13CS016	**Anirudh M Bhat**

students of VIII semester B.E. for the partial fulfillment of the requirements for the Bachelors Degree in Computer Science & Engineering of the **VISVESVARAYA TECHNOLOGICAL UNIVERSITY** during the academic year 2016-17.

Dr.Suneetha K R
Internal Guide
Associate Professor
Department of CSE
Bangalore Institute of Technology
Bengaluru - 560004.

Dr. S. NANDAGOPALAN
Professor and Head
Dept. of Computer Science & Engineering
Bangalore Institute of Technology
K. R. Road, V. V. Puram
Bengaluru - 560004.

Examiners: 1.

2.

“Dedicated to our beloved Parents, Teachers and Friends”

ACKNOWLEDGMENTS

We consider it a privilege to express a few words of gratitude and respect to all those who guided and inspired us in the completion of this project.

We would like to express our sincere gratitude to **Dr. A.G. Nataraj,** Principal of Bangalore Institute of Technology, for providing us a congenial environment and surrounding to work in.

We would like to thank **Dr. S. NANDAGOPALAN**, HOD, Department of Computer Science & Engineering for the infrastructure and facilities provided for the students which helped us in presenting our project in due time.

We are also grateful to our guide **Dr.Suneetha K R**, Associate Professor, Department of Computer science & Engineering for the guidance, without her support it would not have been possible to successfully present this project.

Dwarakanandan B M,
Divyendu Dutta,
Bhargav Chamla,
Anirudh M Bhat

ABSTRACT

Real-time games like Dota 2 lack the extensive mathematical modeling of turn-based games that can be used to make objective statements about how to best play them. Understanding a real-time computer game through the same kind of modeling as a turn-based game is practically impossible. There have been several approaches to determine the outcome of these games. Most approaches use team composition to determine the outcome, whereas a few of them factor into account hero interactions to improve accuracy.

This work proposes to design a model which can predict the match outcome using partial game state data by applying Logistic regression classifier. Outcome may either be Radiant (Team 1) or the Dire (Team 2). Analyze the average accuracy of the model at different game times, and build a Frontend which uses the above model to predict DOTA2 League games.

From the results, it was concluded that partial game-state data can be used to accurately predict the results of an ongoing game of Dota 2 in real-time with the application of machine learning techniques. Graphs were plot to compare the accuracy of different machine learning classification algorithms. And finally a website was developed to predict the outcome of ongoing Dota 2 league games.

LIST OF FIGURES

CONTENTS

INTRODUCTION

1.1 What is a Strategy game?

A **strategy game** or **strategic game** is a game e.g. video or board game in which the players' uncoerced, and often autonomous decision-making skills have a high significance in determining the outcome. Almost all strategy games require internal decision tree style thinking, and typically very high situational awareness.

The term "strategy" comes ultimately from Greek. It differs from "tactics" in that it refers to the general scheme of things, whereas "tactics" refers to organization and execution.

The history of turn-based strategy games goes back to the times of ancient civilizations such as Rome, Greece, Egypt, and India. There were many created and played through their respective regions but only a few have made it to modern day society and are still played. One such game is mancala which was thought to have originated in Samaria approximately 5000 years ago. It spread through the middle east though traders. It challenges two players against each other trying to clear their side of the board of mancala pieces while adding them into their opponent's board to prevent them from clearing their side of the board. There are also two deposits on either side of the board where you must try to deposit the pieces to try and gain points. When one side is cleared the other side of the boards pieces are added to the cleared sides pile often giving them the advantage and allowing them to win. Now mancala is quite a casual game, but it still presents a certain level of strategy by interfering in your opponent's playing area while clearing your own. Another example of one game that stood the test of time is the game of chess. Chess is believed to have originated in India around the sixth century AD. The game was spread to the west once again by trade, but chess became a much greater commodity than many other games of the past. Chess became a game of skill and tactics often forcing the players to think two or three moves ahead of their opponent just to keep up. This game also became a test of intelligence as the some of the smartest people rose to the top and became chess grand masters. The game portrays foot soldiers, knights, rooks, kings, queens, bishops, and rooks each portray actual positions in the military although some moves for some pieces were altered during its travels, it still remains mostly the same. Each

piece has its own special movement pattern, for example the knight is constricted to moving in a pattern the shape of an l that is three long and one block to either side, the rook can only move in a straight line whether it is vertical or horizontal but can move as far as they like so long another piece is notin their way, bishops can move diagonally on the board. It was the strategy and tactics of these games that lead into some of the games created today.

One of today's modern games that has become a sensation for its strategy and tactics is the X-COM franchise, specifically the two most recent games that they released, X-COM: Enemy Unknown (2012) and XCOM 2 (2016). These two games portray the player as a commander of an international organization known as X-COM. The player's job is to repel an alien force using the recourses that you are given by each region and country that is a part of the organization. The game is played through confrontations with the alien force using a squad of 4 to 6 soldiers with periods of time in between where the player is able to even the odds placed against them by upgrading weapons and Armor for the soldiers using technology that is recovered from the aliens. These upgrades result in boosted health as well as laser and plasma based weapons and are necessary to achieve if the player wishes to complete the game. Like chess the games have different classes of soldiers with different abilities which can turn the tide of the game if you use them correctly or not. The come in six classes for each game. In X-COM: Enemy Unknown the soldier classes consist of heavy, capable of dealing heavy damage and carries rockets as well as grenades, the sniper, capable of hitting enemies from beyond line of sight and do immense amounts of damage with single shot, the support, can heal teammates and provide cover using smoke, the assault, which relies on getting up close in order to use the shotgun that they use to make short work of any enemy, the Psionic, this class specializes in applying status effects and generally messing with the opponent's force, and finally the MEC, this used to be a fully organic being but volunteered to replace their organic body with robotic augments, this gives them massive amounts of health and makes them the tanks of the game on the protagonists side.

Another aspect of turn-based strategy rather than just a battlefield in modern video games is controlling countries such as in the civilization franchise and their most recent title Civilization VI. This strategy game forces the player to look at the world as a whole as there are multiple countries involved in the game that will react to the player and their actions and how the influence the world. The player must maintain relations with other

nations as they try to progress their society forward by the inclusion of funding to sections of their society such as mathematics, art, science, and agriculture. Each of these is important to maintain as the player progresses because without the added funds to these branches of society most players will be stuck in the dark ages while other civilizations advance into renaissance eras and further. This can cause turmoil in the player's civilization as well as revolt and will bring the civilization crumbling to the ground. This is only a small portion of the game, the other nations around the player will offer treaties and alliances but some of these are shams and are used to lure the player into a false sense of security as an allied nation begins to take over resources or land that used to belong to the player. In these situations of today's modern games that has become a sensation for its strategy and tactics is the X-COM franchise, specifically the two most recent games that they released, X-COM: Enemy Unknown (2012) and XCOM 2 (2016). These two games portray the player as a commander of an international organization known as X-COM. The player's job is to repel an alien force using the recourses that you are given by each region and country that is a part of the organization. The game is played through confrontations with the alien force using a squad of 4 to 6 soldiers with periods of time in between where the player is able to even the odds placed against them by upgrading weapons and Armor for the soldiers using technology that is recovered from the aliens. These upgrades result in boosted health as well as laser and plasma based weapons and are necessary to achieve if the player wishes to complete the game. Like chess the games have different classes of soldiers with different abilities which can turn the tide of the game if you use them correctly or not. The come in six classes for each game. In X-COM: Enemy Unknown the soldier classes consist of heavy, capable of dealing heavy damage and carries rockets as well as grenades, the sniper, capable of hitting enemies from beyond line of sight and do immense amounts of damage with single shot, the support, can heal teammates and provide cover using smoke, the assault, which relies on getting up close in order to use the shotgun that they use to make short work of any enemy, the Psionic, this class specializes in applying status effects and generally messing with the opponent's force, and finally the MEC, this used to be a fully organic being but volunteered to replace their organic body with robotic augments, this gives them massive amounts of health and makes them the tanks of the game on the protagonists side.

Another aspect of turn-based strategy rather than just a battlefield in modern video games is controlling countries such as in the civilization franchise and their most recent

title Civilization VI. This strategy game forces the player to look at the world as a whole as there are multiple countries involved in the game that will react to the player and their actions and how the influence the world. The player must maintain relations with other nations as they try to progress their society forward by the inclusion of funding to sections of their society such as mathematics, art, science, and agriculture. Each of these is important to maintain as the player progresses because without the added funds to these branches of society most players will be stuck in the dark ages while other civilizations advance into renaissance eras and further. This can cause turmoil in the player's civilization as well as revolt and will bring the civilization crumbling to the ground. This is only a small portion of the game, the other nations around the player will offer treaties and alliances but some of these are shams and are used to lure the player into a false sense of security as an allied nation begins to take over resources or land that used to belong to the player. In these situations, it becomes tricky to navigate as there are two paths, negotiation or war. Often times negotiations are the best choice because it avoids conflict and allows your society to progress further whereas war takes a considerable amount of resources and the player must also be aware of the actual allies that the opposing force has and how much aid they will provide. Unfortunately, negotiations are not always possible and it can result in war, this makes it very important to have loyal allies of your own and a suitable army with sufficient technologies which is all supported by your societies math and science departments respectively**.**

1.2 How are real time strategy games different from turn based strategy games?

One of the strategy genre's most important dividing lines is the manner in which time passes – is it continuous, as in the real world? Or is it segmented into phases designed to restrict player activity? Many strategy fans favour one over the other and the "debates" between these groups often grow contentious. When a prominent series switches sides it often leads to proclamations of imminent doom, or at the very least a fair bit of teeth-gnashing.

While there's certainly been a great deal of conversation pertaining to this topic, rare are truly comprehensive studies which seek to identify what differentiates turn-based

games from their real-time cousins. Good designers need to be well-versed in the strengths and weaknesses of both.

1.2.1 Approachability

One of the most basic differences between the turn-based and real-time mediums is the natural appeal and approachability offered by each. And the issue isn't nearly as simple as *"one type is easy to get into and the other isn't."*

Real-time games offer experiences more akin to everyday life. Sure, waiting in line at the grocery store might be "turn-based," but everything we do is just one link in an endless chain of events. I walk from here to there and it takes a minute or so. Or maybe two. It doesn't really matter. In an RTS you might order some troops across the map, and they'll keep going until they get there and will arrive in a minute or two. This sense of familiarity provides a measure of comfort to many players, particularly those with more casual tastes.

Real-time isn't for everyone though. The time pressure exerted on players can generate feelings of anxiety, sometimes to an extreme degree. Some people relish timed challenges, but many do not.

What should I be doing? Wait, what's that? Oh, I've screwed up already, I just know it. Wow, he found my base already? Ugh, this isn't fun…

Experienced gamers often forget that it takes a fair amount of effort to get into any game more complex than rock-paper-scissors. There are people who do enjoy that initial *"okay, where's the light switch?"* phase, but most would prefer to skip ahead to the *good stuff* – that moment at which they've obtained some level of mastery and are no longer *completely* lost. The ability to learn at one's own pace is a huge plus in turn-based strategy's column.

1.2.2 Pacing

The rate at which events occur is perhaps the biggest difference between turn-based and real-time games. Recall that pacing is simply the rate at which "something interesting" happens. In a turn-based game the designer has virtually no control over when, in terms of actual seconds or minutes, events will take place. A frenetic player could finish a game in an hour – a methodical one might do so in *twenty*.

For many turn-based strategy fans this flexibility is one of the medium's best features, but it's not always a positive. A lesson designers learn early on is that what players think they want and what they *actually* want often don't align in the slightest! One of the rough edges that has long plagued the *Civilization* series is the pacing of the first ten or twenty turns when players only have a couple widgets to fiddle with. The need to hit the enter key five times in a row to get past the boring part is *not* a quality to be proud of. I've watched more than a handful of playtests in which individuals would end their turn only cautiously and reluctantly. And they're right to be hesitant, as a game with better pacing would not have thrust them in such a position.

Designers of a real-time game are blessed with the capacity to know *precisely* that players can train eight space marines in 30 seconds and will have trained their first ultraism between 8 and 12 minutes in. Exact numbers of this sort can never be to *everyone's* liking, but it greatly simplifies the designer's task of ensuring a fairly smooth experience for all.

1.2.3 Prioritization

Turn-based games may not have the advantage of natural pacing, but they certainly make up for it in other ways. Their greatest strength is granting players control in deciding what's important and what isn't.

The manner in which time flows subtly informs players what they should be focusing on. In a broad sense, turn-based games reward analysis and preparation, while real-time ones reward pattern-recognition and execution.

When one is racing against the clock it's more important to be *prompt* than it is to be *perfect*. When an enemy's arrival is imminent, simply putting *any* army into the field takes precedence over tuning its exact composition. As such, real-time games tend to be enjoyed by players who enjoy performing feats of skill and feed off of the mastery developed through practice. With unlimited time it becomes possible to derive the "best" possible solution for a situation. Not everyone's brain works at breakneck speed, and turn-based games offer *everyone* – fast or slow, young or old – the opportunity to exhibit their prowess. While this quality certainly offers advantages, it also comes paired with potential drawbacks.

1.2.4 Micromanagement

The ability, and perhaps, necessity of delving into minutia can be both a weakness (*Civilization3*) and a feature (*StarCraft*). Obviously, the more time players have to make a decision the more of an opportunity there is for them to direct every last detail. This naturally encourages designers of turn-based games to add complexity, and it's possible for these two factors to intertwine and strangle gameplay. *Master of Orion 3* is one such title which strayed way off the deep end.

Real-time games typically feature significantly less micromanagement. By necessity they must hide certain elements behind the scenes, as there is an upper limit to how many balls even the most skillful player can juggle at once. There are also some types of micromanagement that don't really make sense in a real-time game.

The ability to move units between discrete grid tiles is a core aspect of many turn-based games, but trying to wedge such a feature into one that's real-time would be a questionable decision at best. Tiles are an abstraction of the real world which helps designers and players understand and manage the map. A tile-less map is looser and less precise – the *opposite* qualities turn-based games favour. There's a managerial tax associated with tiles that fortunately becomes almost irrelevant when players have unlimited time to make decisions. However, in a real-time game where every second counts do players *really* have time to be worrying about the *specific* plot of land their spearmen are standing on?

When incorporated effectively, micromanagement is an excellent way for players to develop mastery. Both turn-based and real-time games can use it as a tool to highlight the differences in skill between players. The *StarCraft 2* team unabashedly placed an artificially-low cap on the number of units which can be ordered around as a group because they *wanted* an unlevel playing field. Obviously, this approach isn't right for every title, but the success of the *StarCraft* franchise helps remind us that game design is still very much an art where the palette available to developers is vast indeed!

So how do you determine what level of micromanagement is appropriate then? The key factor is usually a game's pacing. A good example of a real-time game that leans more towards the turn-based bucket is Paradox's *Europa Universalise* series. These games offer players many more knobs than a traditional RTS, and their extremely powerful game speed options almost suggest that they're a sort of turn-based/real-time "hybrid." These games offer players many more knobs than a traditional RTS, and their extremely powerful game speed options almost suggest that they're a sort of turn-based/real-time . There are people

who actually play the games as though they were turn-based, pausing the flow of time frequently to survey the situation and issue orders, then resuming for the sole purpose of simulating the “resolution phase” – basically the same flow as in a turn-based game.

1.3 What is Dota2?

Dota 2 is a free-to-play multiplayer online battle arena (MOBA) video game developed and published by Valve Corporation. The game is the stand-alone sequel to defence (DotA), which was a community-created mod for Blizzard Entertainment's Warcraft III: Reign of Chaos and its expansion pack, The Frozen Throne. Dota 2 is played in matches between two teams that consist of five players, with both teams occupying their own separate base on the map. Each of the ten players independently control a powerful character, known as a "hero", that each feature unique abilities and different styles of play. During a match, a player and their team collects experience points and items for their heroes in order to fight through the opposing team's defences. A team wins by being the first to destroy a large structure located in the opposing team's base, called the "Ancient".

Development of Dota 2 began in 2009 when IceFrog, the pseudonymous lead designer of the original defence of the Ancients mod, was hired by Valve to create a modern sequel. Dota 2 was officially released on Steam in July 2013 for Microsoft Windows, OS X, and Linux-based personal computers, following a Windows-only open beta phase that began two years prior. Despite some criticism going towards its steep learning curve and complexity, the game was praised for its rewarding gameplay, production quality, and faithfulness to its predecessor. The game initially used the original Source game engine until it was ported over to Source 2 in 2015, making it the first game to use it. Since its release, Dota 2 has been the most played game on Steam, with peaks of over a million concurrent players. The popularity of the game has led to official merchandise being produced for it, including apparel, accessories, and toys, as well as promotional tie-ins to other games and media. The game also allows for the community to create custom game modes, maps, and cosmetics for the heroes, which are then uploaded to the Steam Workshop.

Dota 2 has a widespread and active competitive scene, with teams from across the world playing professionally in various dedicated leagues and tournaments. Premium Dota 2 tournaments often have prize pools totalling millions of US dollars, the highest of

any eSport. The largest of them is known as The International, which is produced by Valve and held annually at the KeyArena in Seattle. Valve also sponsors smaller, but more frequently held tournaments known as the Majors, which lead up to the International every year. For larger tournaments, media coverage is done by a selection of on-site staff who provide commentary and analysis for the ongoing matches, similar to traditional sporting events. Broadcasts of professional Dota 2 matches are streamed live over the internet, and sometimes simulcast on television networks, with peak viewership numbers in the millions.

1.3.1 Gameplay

Dota 2 is a multiplayer online battle arena (MOBA) video game in which two teams of five players compete to collectively destroy a large structure defended by the opposing team known as the "Ancient", whilst defending their own. As in defence of the Ancients, the game is controlled using standard real-time strategy controls, and is presented on a single map in a three-dimensional isometric perspective. Ten players each control one of the game's 113 playable characters, known as "heroes", with each having their own design, benefits, and weaknesses. Heroes are divided into two primary roles, known as the "carry" and "support". Carries, which are also called "cores", begin each match as weak and vulnerable, but are able to become more powerful later in the game, thus becoming able to "carry" their team to victory. Supports generally lack abilities that deal heavy damage, instead having ones with more functionality and utility that provide assistance for their carries.

All heroes have a basic damage-dealing attack, in addition to powerful abilities. Each hero has at least four abilities, all of which are unique, which are the primary method of fighting.The most powerful ability for each hero is known as their "ultimate", which requires them to have an experience level of six in order to learn and use. In order to prevent abilities from being spammed without consequence, a magic system in the game exists. Activating an ability costs a hero some of their "mana points", which regenerate slowly over time.Using an ability will also cause it to enter a cooldown phase, in which the ability can not be used again until a timer counts down to zero. All heroes have three attributes: strength, intelligence, and agility, which affect health points, mana points, and attack speed, respectively. Each hero has one primary attribute out of the three, which adds to their non-ability basic damage. Heroes begin each game with an experience level of one, only having access to one of their abilities, but are able to level up and become more powerful during the course of the game, up to a maximum level of 25. Whenever a hero gains an experience

level, the player is able to unlock another of their abilities or improve one already learned. Heroes also have an ability augmentation system known as "Talent Trees", which allow players further choices on how to develop their hero. If a hero runs out of health points and dies, a timer begins to count down until they respawn in their base.

The two teams—known as the Radiant and Dire—occupy fortified bases in opposite corners of the map, which is divided in half by a crossable river and connected by three paths, where are referred to as "lanes".The lanes are guarded by defensive towers that slowly, but frequently, attack any opposing unit who gets within its line of sight. A small group of weak computer-controlled creatures called "creeps" travel predefined paths along the lanes and attempt to attack any opposing heroes, creeps, and buildings in their way. The two teams—known as the Radiant and Dire—occupy fortified bases in opposite corners of the map, which is divided in half by a crossable river and connected by three paths, where are referred to as "lanes".Creeps periodically spawn throughout the game in groups from two buildings, called the "barracks", that exist in each lane and are located within the team's bases. The map is also permanently covered for both teams in fog of war, which prevents a team from seeing the opposing team's heroes and creeps if they are not directly in sight of an allied unit. The map also features a day-night cycle, with some features and mechanics being altered depending on the time of the cycle.

Items known as "wards" are able to be placed in most locations on the map, granting line of sight vision in a small area around it for the team who had planted it. Wards last for six minutes after being placed, and will disappear once time runs out or if discovered and destroyed by the opposing team. Also present on the map are "neutral creeps" that are hostile to both teams, and reside in marked locations on the map known as "camps". Camps are located in the area between the lanes known as the "jungle", which both sides of the map have. Neutral creeps do not attack unless provoked, and will respawn over time if killed. Items known as "wards" are able to be placed in most locations on the map, granting line of sight vision in a small area around it for the team who had planted it. Wards last for six minutes after being placed. The most powerful neutral creep is named "Roshan", who is a unique boss that may be killed by either team to obtain an item that allows a one-time resurrection by the hero that holds it. Roshan will respawn around ten minutes after being killed, and becomes progressively harder to kill as the match continues over time.

In addition to having abilities becoming stronger during the game, players are able to buy items that provide their own special abilities. Items are not limited to specific heroes,

and can be bought by anyone. In order to obtain an item, players must be able to afford it with gold, which is primarily obtained by killing enemy heroes, destroying enemy structures, and killing creeps, with the latter being an act called "farming". Only the hero that lands the killing blow on a creep obtains gold from it, an act called "last hitting". Players are also able to "deny" allied units and structures by destroying them, which then prevents their opponents from getting full experience. Gold can not be shared between teammates, with each player having their own independent stash. The player also receives a small, continuous stream of gold over the course of a match.

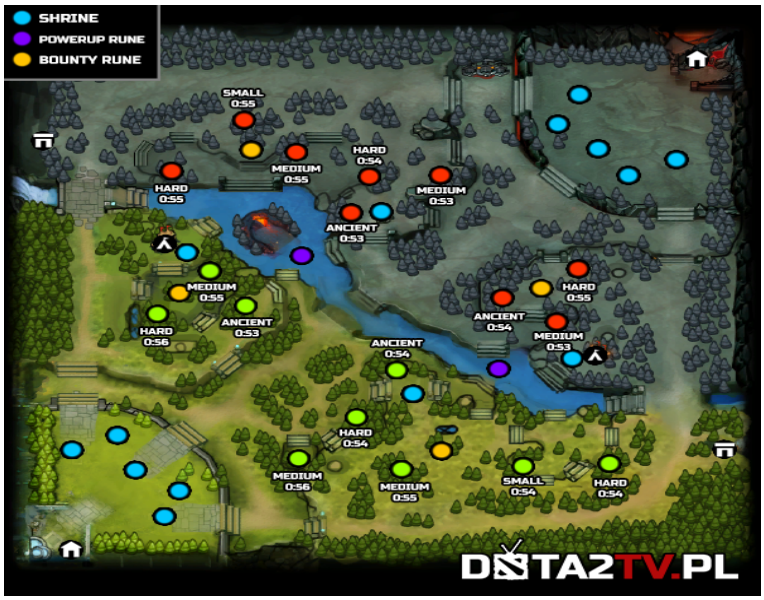


Fig 1.1 Map of Dota 2

LITERATURE SURVEY

Dota 2 is a hugely popular online game and over time has attracted a lot of attention regarding its predictive analysis due to the large prize pools in competitive Dota 2 games, easily available match data etc. There have been several approaches to determine the outcome of real time strategy games like Dota 2. Most approaches use team composition to determine the outcome [4][5], whereas a few of them factor into account hero interactions to improve accuracy [1][2][4].

2.1 Focus on team composition

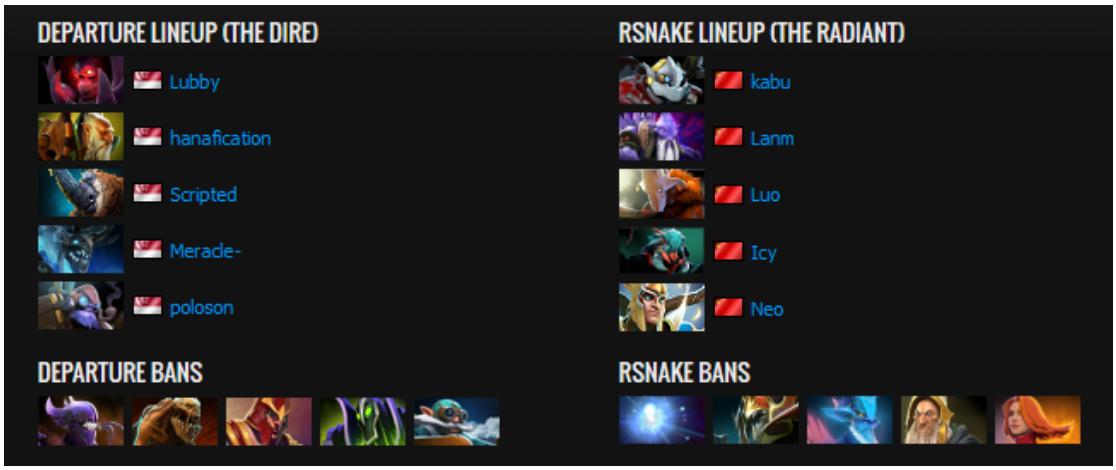


Fig 2.1 Hero composition

In Dota2 each team consists of 5 players and each player controls a specific hero which he/she can select from a roster of 113 available heroes. Team composition refers to which heroes make up a particular team ie, either radiant or dire. Some approaches just consider this information to predict the match outcome. These approaches effectively create a feature vector with value of V_i as either 1 or -1 or 0 corresponding to whether the hero i was in radiant or dire or otherwise. Classification algorithms like logistic regression are then used to predict the match outcome[1][2][5]

There are some major drawbacks of such approaches:

1. Team composition on its own is not a good way of predicting the match outcome as its dosent take into account how the player uses the hero in the game since different players have different skill level with heroes.
2. The approach comes into picture only during the drafting stage ie when the players are selecting their heroes and once the match actually begins, these approaches don't make use of any information from within the game.
3. Heroes are subject to change in future patches of the game making the approach obselete

2.2 Factoring into account hero interactions

As stated above just considering hero/team composition is not a good way of predicting the match outcome and has low accuracy. Hence some approaches use hero interactions to improve the accuracy of their prediction. Hero interactions are basically how the different heroes compliment or counter each other via their abilities. A graph based algorithm can be used to determine the most successful set of heroes ie, which heroes when combined with certain other types of heroes increases the win probability of the team [2]. These successful set of heroes are determined based on how they can compliment each other via their abilities. This is also known as synergistic behaviour.

From publically available match data, the win rate of each hero can be found out. This forms the baseline predictor. Combining this with hero interactions in the form of synergy and countering increases the accuracy of prediction [4].

- Synergy is how abilities of heroes can be used to compliment each other and help the team gain advantage.
- Countering is how the abilities of one hero in a team can counter the abilities of other heroes in the opponent team.

Another major approach to factoring hero interaction is via hero combos. Hero combo is basically a chain of abilities of two or more heroes in a team. There are a lot of

heroes and an even greater number of hero combos, so the fifty most powerful 2 hero combo is a good start [1].

Fig 2.2 Two hero combo

Some approaches also take into account hero items which is a way of considering hero interaction since heroes may utilize their items synergistically to their advantage [3]. Hero items are equipment that can be bought in the game that provide heroes with special attributes and abilities. Lower tier items can further be combined to high tier items. Players have six item slots in their inventory. The drawback of considering hero items is the same as hero composition that is, items get updated in future patches and will make the approach to fail in the future.

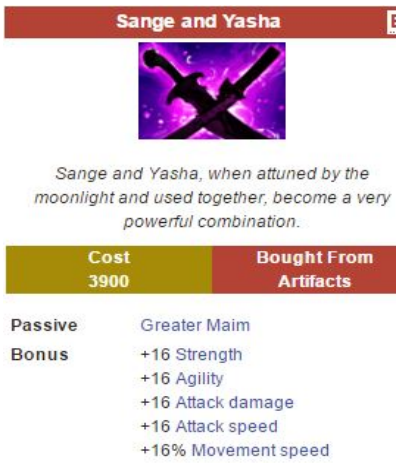


Fig 2.3 The “sange and yasha” item and its stats

2.3 Comparison of classification algorithms

Different classification algorithms have been compared. Specifically the accuracy of the random forest classification algorithm against the other classification algorithms is checked [3]. The random forest classification algorithm is determined to have the highest accuracy. Random forest uses a multitude of decision trees for classification. Each tree votes for a particular class and the algorithm chooses the class having the highest votes when classifying. Average accuracy of 25 different random forest models using different values for number of trees and other attributes is compared as well.

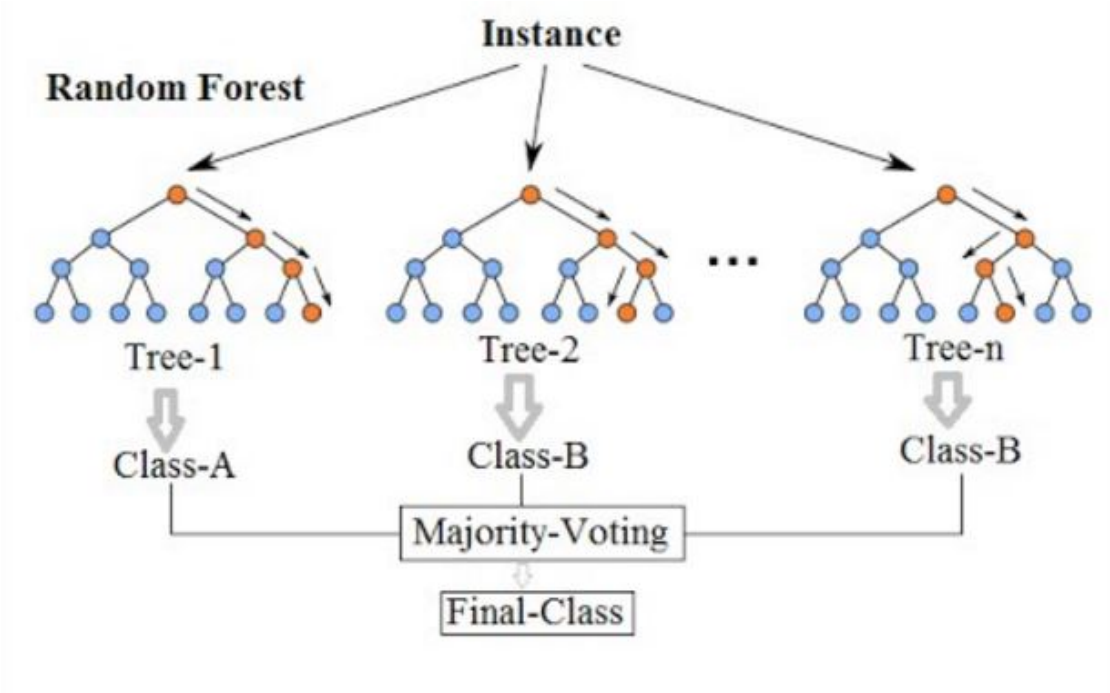


Fig 2.4 Random forest classification algorithm

A lot of approaches make use of logistic regression to predict the match outcome[1][2][5]. The features are selected based on domain knowledge and some exploratory analysis such as gold per minute, XP per minute, kills, deaths etc.[]. XP per minute and Gold per minute vary with heroes. Usually its seen that a hero with high XP per minute has high Gold per minute and on the other end of the spectrum a heroe with low XP per minute has low Gold per minute. Carry heroes typically are at the top of such lists whereas support heroes are at the bottom of these lists.

Kills per death also vary with heroes. At the top are Ganker heroes and at the bottom are hard support heroes. A good team should have a combination of all types of heroes so that each hero can support other heroes. XP per minute and Gold per minute are not indicators of hero strength but rather hero role.

2.4 Drawbacks of existing work

All previous work which heavily depended on hero compositions have the drawback that their model dosen't take into account the player himself. So a particular player may use a hero more skilfully than another player but this information is not taken into account by these models. They consider all players to be using the heroes with the same level of skill. Also these models come into picture during the "drafting stage" that is, when the players are selecting their heroes before the match starts. But once the match actually begins their model no longer comes into picture. It does not use any information from within the game itself.

The other drawback is that some models make use of hero items which just like heroes themselves are subject to changes in the future patches of the game making the model irrelevant in the future. These models would constantly be needed to train with fresh data and all existing training data becomes useless since they don't contain information about the updates made in the game.

All these models could have improved accuracy but just relying on data available before the game starts and not using any in game data dosent't allow them to reach the highest possible accuracy.

2.5 Proposed work

The work proposed intends to predict the outcome at each minute of the game which effectively utilizes the player data,match data and summarizes the game state at any point of time. The model being built is independent of hero composition and hero items, keeping it relevant even in the future. And finally a website which makes use of the model as a backend to predict the outcome of ongoing Dota2 league games is to be developed.

SYSTEM REQUIREMENTS

3.1 Hardware requirements

- State of the art system

3.2 Software requirements

- Functional Requirements
  An appropriate Machine Learning Classifier, trained with a large reference game database.
- Usability - An automated prediction tool which is easy to use without any human intervention.
- Performance - The tool should be faster in execution. Lesser Spatial Complexity.
- Operating System – Windows 7 or above, Linux (Ubuntu, Linux Mint).
- Java Runtime Environment 7 or above
- Python 2.7 with Numpy, Scikit-Learn, Matplotlib modules
- Node JS, modules for Steam and Dota 2
- Git and Git-Hub for version control.

3.3 Assumptions

The following assumptions have been made in this project:

- Storage space is not an issue. Our model uses a machine learning and as such would require a lot of training data.
- We have only used game data which have 5 players in each team since that is the standard in competitions.
- Only matches with game duration more than 30 minutes are chosen since they will have enough data by then to effectively characterize that match.

ARCHITECTURE

4.1 Problem definition

Construct a model which can predict the match outcome using partial game state data, using various data mining and machine learning techniques. Outcome may either be Radiant (Team 1) or the Dire (Team 2). Also, Analyse the average accuracy of the model at different game times, and build a Frontend which uses the above model to predict DOTA2 League games.

We make us game state data i.e. the values of various variables that may affect the games progress using various data mining techniques and once we obtain this state data for each minute we train various machine learning models.

We make use of this model to predict an ongoing match with its available state data, the output of the model is either radiant or dire (i.e. one of the two teams can be the winner). Since we have state information at different times of the game we analyse the average accuracy of the model at different game times.

We build a frontend, here a website which showcases the probable winner of the ongoing matches with the help of our trained model. The frontend is in the form of a website which first displays details of ongoing matches and when clicked shows the prediction details of that particular match.

As mentioned earlier Dota2 is a real-time strategy game played by utmost 10 players, these players have 112 heroes to pick from and limitless items to equip these heroes and every combination of hero and item has a different kind of impact on the ongoing game.

.

4.2 Proposed Architecture

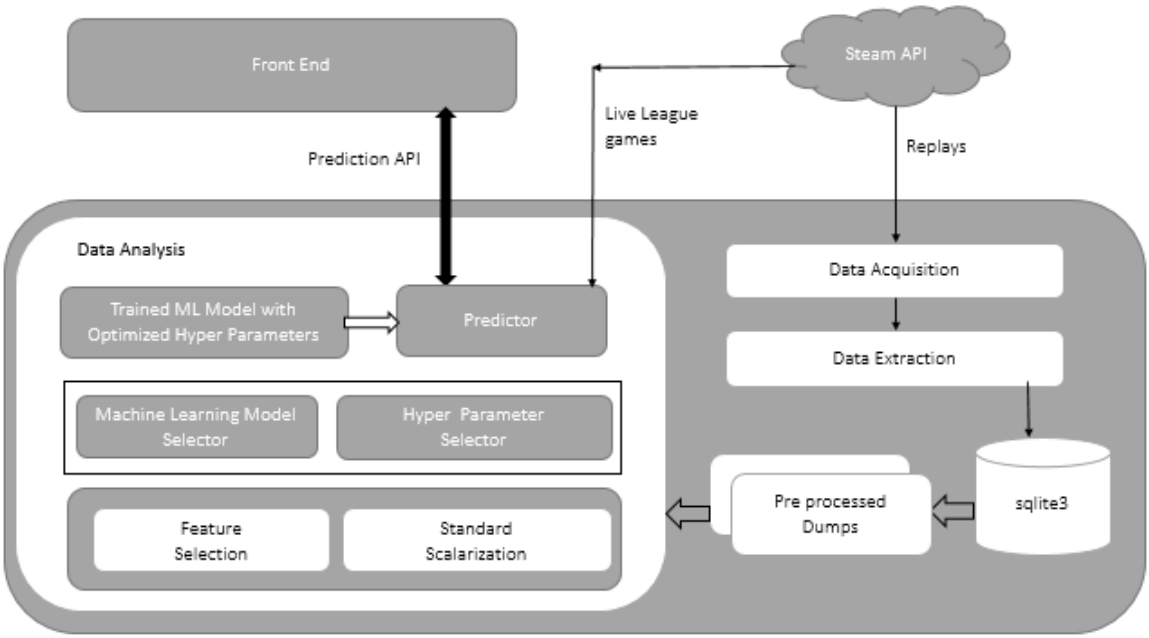


Fig 4.1 Overall architecture

The major goal of this architecture is to give an output either as radiant or the dire i.e. predicting which team has a higher chance of winning at this moment of time, for the system to do that, we begin with downloading the replays from steam and the following steps or modules will take us to the desired output,

4.2.1 Data Acquisition

This step involves fetching the replay files from the Steam servers. Firstly a script retrieves a list of match-ids which meet the constraints, each match is represented as a JSON file. The game client is emulated to obtain the URL for the replay using the JSON data. The URL is constructed using cluster ID, match ID and replay salt for every match. A plugin called Node-Dota2 is used to emulate the Dota2 Client. The replays obtained are in the form of demo files which are compressed using bz2 (*.dem.bz2). These replays are un-compressed and checked for integrity.

4.2.2 Data Extraction

This step involves extracting data from the demo files and building a SQLite database to store all the relevant attributes. Replay files mainly consist of network packet information which were exchanged between the game client and server encoded in Google's Proto-Buff format. A plugin called Clarity-2 will be used to extract this raw data. Schema for the database consists of two tables. A MATCH_DATA table which mainly has match-ids and their outcomes along with other related attributes. A PLAYER_DATA table which hold player information at every minute of the game. Data for the first 30 minutes of each game is recorded here. A total of 170 attributes are recorded, 17 attributes per player. This database is the primary data source for subsequent steps of this project.

4.2.3 Data Analysis

This step involves building the machine learning model which will be used to perform the prediction. The data analysis is done mainly in Python using its wealth of plugins and modules. Numpy, Scipy, Scikit-Learn, Matplotlib to name a few.A feature extraction step is performed to identify the best set of features. Since this is a classification problem, more specifically a binary classification problem, a set of feasible classification algorithms was identified, such as Logistic Regression, K nearest neighbours, Random Forest Classifier and Support Vector Machine. Grid Search Cross Validation is used to identify which algorithm works best for our dataset. Hyper Parameter tuning is performed on these algorithms to obtain the best accuracy.Logistic Regression had the highest accuracy and so it was chosen for the machine learning model. The highest accuracy achieved was achieved with Logistic Regression Classifier at 30 min, which is 97.4%.

4.2.4 Front-End (Website)

Steam provides a Web API to obtain live league game data. We made use of Django which is a free and open source web framework written in python to build our website. This uses our backend machine learning model to predict the winner of an ongoing Dota2 league match.

SYSTEM DESIGN

5.1 Flow chart

Our system is made up of three major modules and the at the end we make use of a website as the user interface for the model. The flow of our system is shown in figure 5.1.1

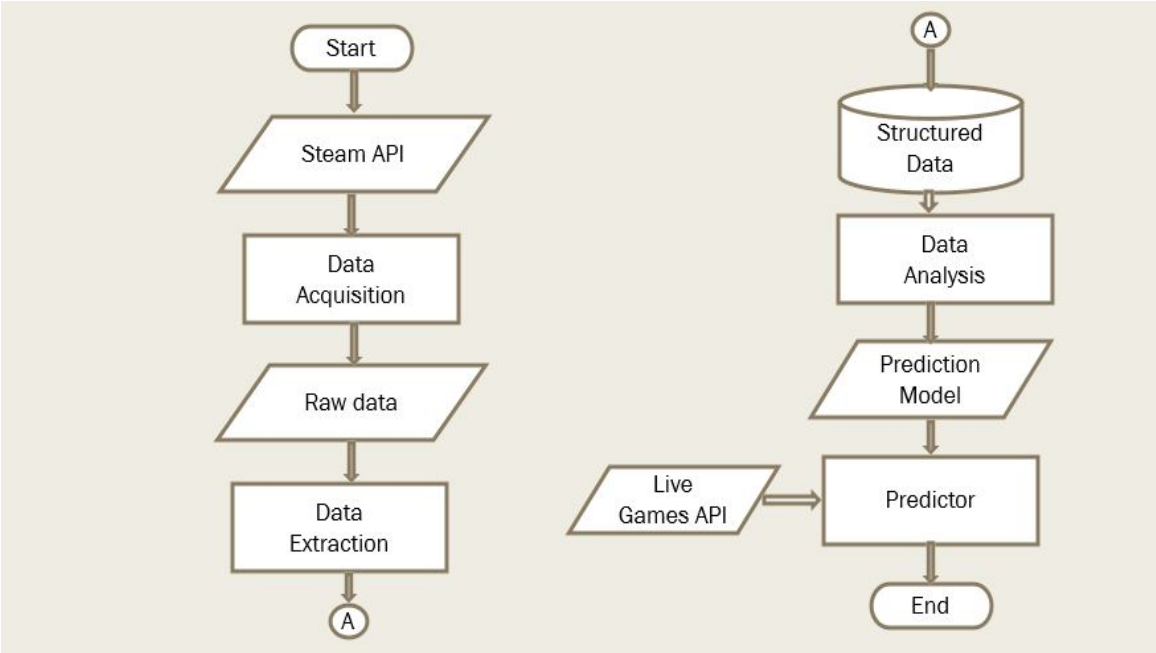


Fig 5.1 Flow chart

In the flow chart we see, the raw data that we are downloading is being provided by steam and these replays can be accessed through their api. We build urls for each replay we download using the requisite parameters and store the replays in .dem format at the end of our first module.

The second module also called the data extraction phase makes use of a clarity parser to extract the required features from the obtained .dem files and stores them in a structured format in a database. This data that we store in the structured data base is used to train various machine learning algorithms such as SVM, Random forest, Logistic regression. We make use of these trained models to predict the live games whose state data is provided through another steam's api which is one of the final goal of this system.

The detailed description of each module including the frontend i.e. the website is provided in the following sections.

5.2 Module Description

The four major modules which comprises of the entire system are explained in a detailed manner here. The four modules that are being explained are data acquisition, data extraction, data analysis and the frontend.

5.2.1 Data Acquisition

Data acquisition is the foremost thing that must be done in order for us to perform the predictive analysis of the real-time strategy game dota2.There is no special dataset that's been made available for us to make use of, hence for our model we consider building our own dataset consisting of the match details.

The replay files of the matches are available on the steam databases and for us to be able to download them we need to construct a unique url for every replay, and to construct this url is not a straight forward process, few values and keys must be procured for every replay such that the url can be constructed.

5.2.1.1 Algorithm for data acquisition

```
N <= Number of replays to download
C <= Steam Credentials to use while downloading
match_ids <= null
While (length of match_ids < N) do:
        list_match_ids <= Obtain Match ID's of last 100 matches from Steam API using
                                credentials C
        For (i=0 to 100) do:
                len <= length of match i
                if( len > 30min) then:
                        Append current match to match_ids
                else:
                        Discard current match
        end for
End while

Args <= build arguments to be passed to the nodejs dota client
```

Call the Dota2 Client

For (i=0 to N) do:

    Download the match details and store it in JSON format

End for

N_Threads <= Set the number of threads to be used

Split the match ID's equally among the N threads

For each thread do:

    For (i=0 to match ID's assigned to current thread) do:

        Construct download URL based on match_id, replay_salt, cluster_id

        Download the Match Replay using the match details and store it in *.dem.bz2

        Extract the *.bz2 files and check match replay for integrity

    end for

End for

5.2.1.2 main_handler.py

The basic requirement for us to construct the url is for us to obtain the match id's, and steam doesn't let everyone to download the matches as it incurs extra costs for unnecessary downloads. Hence a developer key must be procured by making a prime account with steam.

Once this key is available we gain access to a private api bearing the url- *"https://api.steampowered.com/IDOTA2Match_570/GetMatchDetails/V001/?match_id="+str(match_id)+"&key=ACABEB1FD8894A44B2A5AB4B79209C75"*, this url gives us the json files for the latest 100 matches per account per day.

The json files that are present now are not of much use as they present the end game values and they also don't hold all the necessary values required to construct the url that is needed to download the entire match replay.We parse these json and take only two entities i.e. match id's and match durations, we take match duration only to make sure we download matches with duration above 30 mins.

A value called the replay salt that is unique to every replay is not available to us from these replays and the replay salt is an essential to complete the url for the download. For us to get the replay salt we have to be in game and access the replays from in game,

since this is not a viable option we emulate the dota2 client using node.js. After the client is emulated we get another json file for every replay that looks like this,

Match Details JSON


```
{
  "result":1,
  "match":{
    "duration":1892,
    "startTime":1492161292,
    "cluster":156,
    "first_blood_time":174,
    "replay_salt":771916548,

    …...
  },
  "vote":0
}
```


The work of main handler ends at the moment we get these json files all the match id's we have in possession. The Dota2 client is emulated as a sub process from within the main handler itself.

The main_handler.py when executed gives the following output,

$ python main_handler.py

1 Selected Match_id: 3117261373 Duration: 1937 seconds

2 Selected Match_id: 3117260732 Duration: 1892 seconds

3 Selected Match_id: 3117260232 Duration: 1827 seconds

Calling dota client

14 Apr 15:22:52 - Launching Dota 2

14 Apr 15:22:53 - Sending ClientHello

14 Apr 15:22:59 - Received client welcome.

14 Apr 15:22:59 - Sending match details request

14 Apr 15:23:00 - Received match data for: 3117261373

1 Got match details - 3117261373

14 Apr 15:23:00 - Sending match details request

14 Apr 15:23:00 - Received match data for: 3117260732

2 Got match details - 3117260732

14 Apr 15:23:00 - Sending match details request

14 Apr 15:23:01 - Received match data for: 3117260232

3 Got match details - 3117260232

Finished downloading

The above script was set to download 3 replays hence its obtained 3 json's for each replay.

5.2.1.3 download _handler.py

All that's been done till now is just gathering the values and variables needed to build the url that's unique to each replay which downloads the entire replay in .dem.bz2 format. The url that's unique to every replay is built as follows,

"http://replay"+str(cluster)+".valve.net/570/"+match_id+"_"+str(replay_salt)+".dem.bz2".

The cluster id and the replay salt are obtained from the json file corresponding to the match id that has to be downloaded.The value 570 present in the url is the Dota2 equivalent for the valve servers. We need the cluster id as the replays are stored in different clusters, and as mentioned earlier the replay salt is needed to make sure that the steam servers don't have to bear with unnecessary downloads which results in extra load to the servers.

Once the urls are built for the particular match id its appended to a list, and once a hundred such urls are built for every match id, we download the entire replay which has the values for all the attributes of the game for every tick (metric in which time is measured in the game replays 30 ticks=1 min). To maximize the efficiency of the download and make use of the entire bandwidth while downloading the replays we make use of threads to download multiple replays at the same time. We download the replay files in chunks of 1024 bytes.

The downloaded files are in the form of .dem.bz2 format and for our clarity parser to work(parse the replay files to construct the database) we need the files in .dem format, hence we run another script called the extract.py to get the uncompressed .dem files from the .dem.bz2 files

$ python download_handler.py

Starting Thread 0

Starting Thread 1

[Thread 1]Downloading http://replay156.valve.net/570/3117260732_771916548.dem.bz2

[Thread 0]Downloading http://replay152.valve.net/570/3117260232_1224521506.dem.bz2

Starting Thread 2

[Thread 2]Downloading http://replay186.valve.net/570/3117261373_2087911471.dem.bz2

Exiting Thread 2

Exiting Thread 0

Exiting Thread 1

Exiting Main Thread

$ python extract.py

Starting Thread 0

[Thread 0]Extracting 3117260232_1224521506.dem.bz2

Starting Thread 1

[Thread 1]Extracting 3117260732_771916548.dem.bz2

Starting Thread 2

[Thread 2]Extracting 3117261373_2087911471.dem.bz2

Exiting Thread 0

Exiting Thread 1

Exiting Main Thread.

5.2.2 Data Extraction

The data extraction module uses the demo files or ".dem" files to extract data from them and store them in a SQLite database. Dota is a real time multiplayer game meaning it is not turn based. The most fundamental unit of time in real time multiplayer games is a tick.

Informally a tick is the same as a clock cycle or the smallest unit of time recognized by the server. Dota 2 runs at a tick rate of 30 ticks/second. This translates to the fact that there are 30 interactions between the server and every client.

5.2.2.1 Algorithm for data extraction

```
match_ids <= List of match_ids to be parsed
Spawn N worker threads and distribute the match_ids among them
for each worker thread do:
    for (i=0 to number of match id's assigned to thread) do:
        if (current match not in database) then:
            Replay <= Load the replay file of the match
            while (First rune not spawned) do:
                Increment Replay ticks
            end while
            db_thread <= obtain thread level database connection
            populate the MATCH_DETAILS table
            for (min 1 to 30) do:
              populate PLAYER_DATA table for the current match at ith minute
            end for
            close thread level database connection
        end if
    end for
end for
```

5.2.2.2 Protocol Buffers

Each of these interactions are exchange of google's protocol-buffer packets. Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data similar to XML, but smaller, faster, and simpler. Users define how they want their data to be structured once, then they can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages. Data structures can even be updated without breaking deployed programs that are compiled against the "old" format.

```
message Person {
  required string name = 1;
  required int32 id = 2;
  optional string email = 3;

  enum PhoneType {
    MOBILE = 0;
    HOME = 1;
    WORK = 2;
  }

  message PhoneNumber {
    required string number = 1;
    optional PhoneType type = 2 [default = HOME];
  }

  repeated PhoneNumber phone = 4;
}
```

Fig 5.2 Example of protocol buffer message definition

Serialization is the process of translating data structures or object state into a format that can be stored and reconstructed later in the same or another environment. The structure of the information that is to be serialized is specified by the developers by defining protocol buffer message types in .proto files. The incoming data stream is structured based on this definition then serialized, encoded and transmitted over the network. So the demo files contain the information present in every packet that is transmitted by the server.

Protocol buffers have many advantages over XML for serializing structured data.
Protocol buffers:

- are simpler
- are 3 to 10 times smaller
- are 20 to 100 times faster
- are less ambiguous
- generate data access classes that are easier to use programmatically

5.2.2.3 Clarity

A Java based community developed parser called clarity is used to parse the replays. This parser can provide the in game information about various aspects of the game at each and every tick of the game such as:

- **combat log**: a detailed log of events that happened in the game.
- **entities**: in-game things like heroes, players, and creeps.
- **modifiers**: auras and effects on in-game entities.
- **temporary entities**: fire-and-forget things the game server tells the client about.
- **user messages**: many different things, including spectator clicks, global chat messages, overhead events (like last-hit gold, and much more), particle systems, etc.
- **game events**: lower-level messages like Dota TV control (directed camera commands, for example), etc.
- **voice data**: commentary in pro matches.
- **sounds**: sounds that occur in the game.
- **overview**: end-of-game summary, including players, game winner, match id, duration, and often picks/bans

Clarity uses an event based approach to analyse replays. We use it by supplying it with one or more processor which is a POJO (Plain Old Java Object). A simple implementation generally involves:

- Creating a source from the replay file.

- Create a runner with the source. The runner is what drives the replay analysis. A simple runner will run over the whole replay once. A more sophisticated ControllableRunner uses a separate thread for doing the work and also allows seeking back and forth in the replay
- Creating an instance of the processor.
- Run with the processor

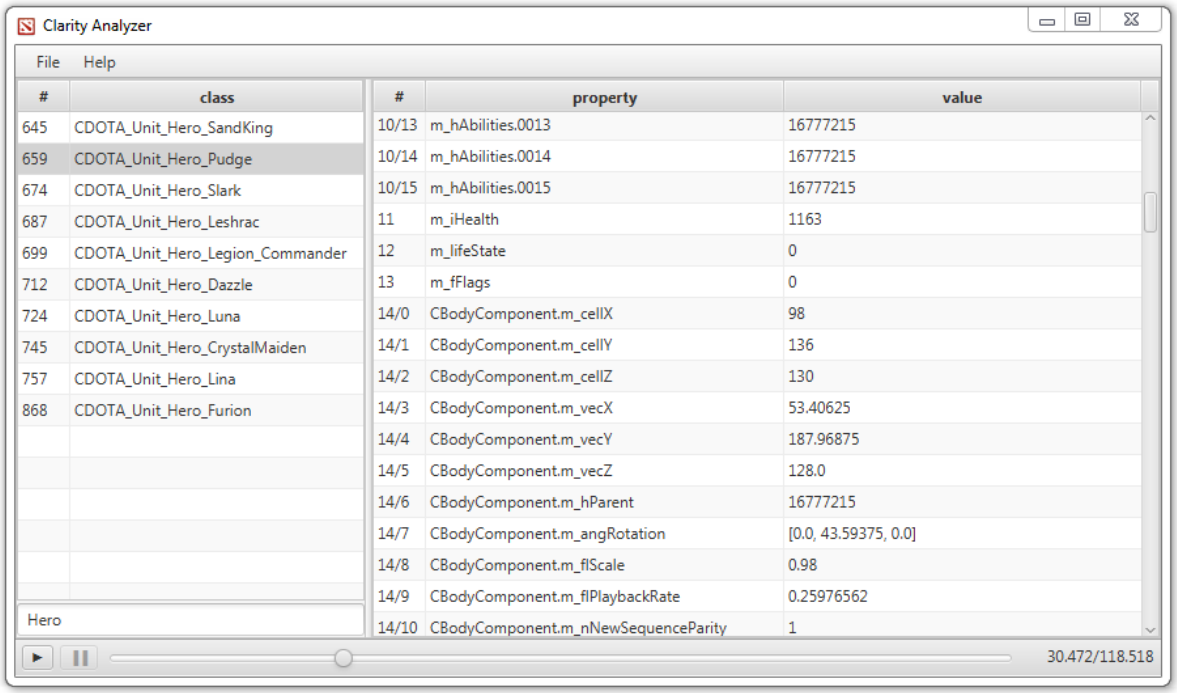


Fig 5.3 An image of clarity GUI

5.2.2.4 Apache Maven

Clarity has various dependencies which are needed for it to run without any problems. All the dependencies are handled in an efficient manner by Apache Maven. Apache Maven is a software project management and comprehension tool that can be used for building and managing any Java-based project. Maven can manage a project's build, reporting and documentation from a central piece of information.

Some of the features provided of Maven are:

- Simple project setup that follows best practices
- Consistent usage across all projects means no ramp up time for new developers coming onto a project
- Superior dependency management including automatic updating, dependency closures
- Able to easily work with multiple projects at the same time
- A large growing repository of libraries and metadata to use out of the box
- Extensible with the ability to easily write plugins in Java or scripting languages
- Instant access to new features with little or no extra configuration
- Model based builds: Maven is able to build any number of projects into predefined output types such as a JAR, WAR, or distribution based on metadata about the project, without the need to do any scripting in most cases

Our model uses data of every minute from the first minute of the game till the 30th minute. Our database schema includes two tables, namely MATCH_DATA and PLAYER_DATA. The MATCH_DATA table consists of details about the match(attributes) such as match ID, game mode, duration, match outcome and every player's hero IDs. PLAYER_DATA consists of details(attributes) regarding each player's hero such as level, kills, deaths etc.

```
DWMOHAN-M-C2HZ:resources dwmohan$ sqlite3 dota2.db
SQLite version 3.8.10.2 2015-05-20 18:17:19
Enter ".help" for usage hints.
sqlite> .tables
MATCH_DATA   PLAYER_DATA
sqlite>
```

Fig 5.4 Tables used

Each match consists of one tuple in MATCH_DATA and thirty tuples per match, one tuple for each minute of the match from minutes 1 to 30 in PLAYER_DATA. The primary key of MATCH_DATA is match_id which is unique for each match. match_id of PLAYER_DATA is a foreign key which references match_id of the MATCH_ID. match_id along with time_elapsed is the composite primary key of PLAYER_DATA. A total of 170 attributes are recorded in PLAYER_DATA table.

```
sqlite> .schema MATCH_DATA
CREATE TABLE MATCH_DATA(MATCH_ID INT PRIMARY KEY, GAME_MODE INT, DURATION I
NT, MATCH_OUTCOME TEXT, PLAYER_0_HERO_ID INT, PLAYER_1_HERO_ID INT, PLAYER_
2_HERO_ID INT, PLAYER_3_HERO_ID INT, PLAYER_4_HERO_ID INT, PLAYER_5_HERO_ID
 INT, PLAYER_6_HERO_ID INT, PLAYER_7_HERO_ID INT, PLAYER_8_HERO_ID INT, PLA
YER_9_HERO_ID INT);
sqlite>
```

Fig 5.5 Schema of MATCH_DATA table

```
sqlite> .schema PLAYER_DATA
CREATE TABLE PLAYER_DATA (MATCH_ID INT, TIME_ELAPSED INT, PLAYER_0_ASSISTS
INT, PLAYER_0_CAMPS_STACKED INT, PLAYER_0_DEATHS INT, PLAYER_0_DENY_COUNT I
NT, PLAYER_0_HEALING REAL, PLAYER_0_KILLS INT, PLAYER_0_LAST_HIT_COUNT INT,
 PLAYER_0_LEVEL INT, PLAYER_0_NET_WORTH INT, PLAYER_0_OBSERVER_WARDS_PLACED
 INT, PLAYER_0_ROSHAN_KILLS INT, PLAYER_0_RUNE_PICKUPS INT, PLAYER_0_SENTRY
_WARDS_PLACED INT, PLAYER_0_STUN_DURATION REAL, PLAYER_0_TOTAL_GOLD_EARNED
INT, PLAYER_0_TOTAL_XP_EARNED INT, PLAYER_0_TOWER_KILLS INT, PLAYER_1_ASSIS
TS INT, PLAYER_1_CAMPS_STACKED INT, PLAYER_1_DEATHS INT, PLAYER_1_DENY_COUN
T INT, PLAYER_1_HEALING REAL, PLAYER_1_KILLS INT, PLAYER_1_LAST_HIT_COUNT I
NT, PLAYER_1_LEVEL INT, PLAYER_1_NET_WORTH INT, PLAYER_1_OBSERVER_WARDS_PLA
CED INT, PLAYER_1_ROSHAN_KILLS INT, PLAYER_1_RUNE_PICKUPS INT, PLAYER_1_SEN
TRY_WARDS_PLACED INT, PLAYER_1_STUN_DURATION REAL, PLAYER_1_TOTAL_GOLD_EARN
ED INT, PLAYER_1_TOTAL_XP_EARNED INT, PLAYER_1_TOWER_KILLS INT, PLAYER_2_AS
SISTS INT, PLAYER_2_CAMPS_STACKED INT, PLAYER_2_DEATHS INT, PLAYER_2_DENY_C
OUNT INT, PLAYER_2_HEALING REAL, PLAYER_2_KILLS INT, PLAYER_2_LAST_HIT_COUN
T INT, PLAYER_2_LEVEL INT, PLAYER_2_NET_WORTH INT, PLAYER_2_OBSERVER_WARDS_
PLACED INT, PLAYER_2_ROSHAN_KILLS INT, PLAYER_2_RUNE_PICKUPS INT, PLAYER_2_
SENTRY_WARDS_PLACED INT, PLAYER_2_STUN_DURATION REAL, PLAYER_2_TOTAL_GOLD_E
ICKUPS INT, PLAYER_9_SENTRY_WARDS_PLACED INT, PLAYER_9_STUN_DURATION REAL,
PLAYER_9_TOTAL_GOLD_EARNED INT, PLAYER_9_TOTAL_XP_EARNED INT, PLAYER_9_TOWE
R_KILLS INT, PRIMARY KEY (MATCH_ID,TIME_ELAPSED), FOREIGN KEY (MATCH_ID) RE
FERENCES MATCH_DATA(MATCH_ID) ON DELETE CASCADE);
sqlite>
```

Fig 5.6 Schema of PLAYER_DATA table

We use multithreading to use 4 worker threads to efficiently parse the replays. The replays that were extracted at the end of the previous step are stored using the naming format “matchid_replaysalt.dem”. A connection to the main database is obtained using the sqlite JDBC driver. Using this connection, a list of all the matchids present in the main database is stored in an array list. The list of all the file names in the directory containing the demo files are passed to each of the worker threads along with the array list of matchIds that are currently present in the database.

Each of the worker threads performs some initialization of variables followed by obtaining a thread level connection to an intermediate database based on the thread ID. For

each of the filenames passed to the thread, it checks if that the matchid from the file name already exists in the database and filters the already parsed replays.

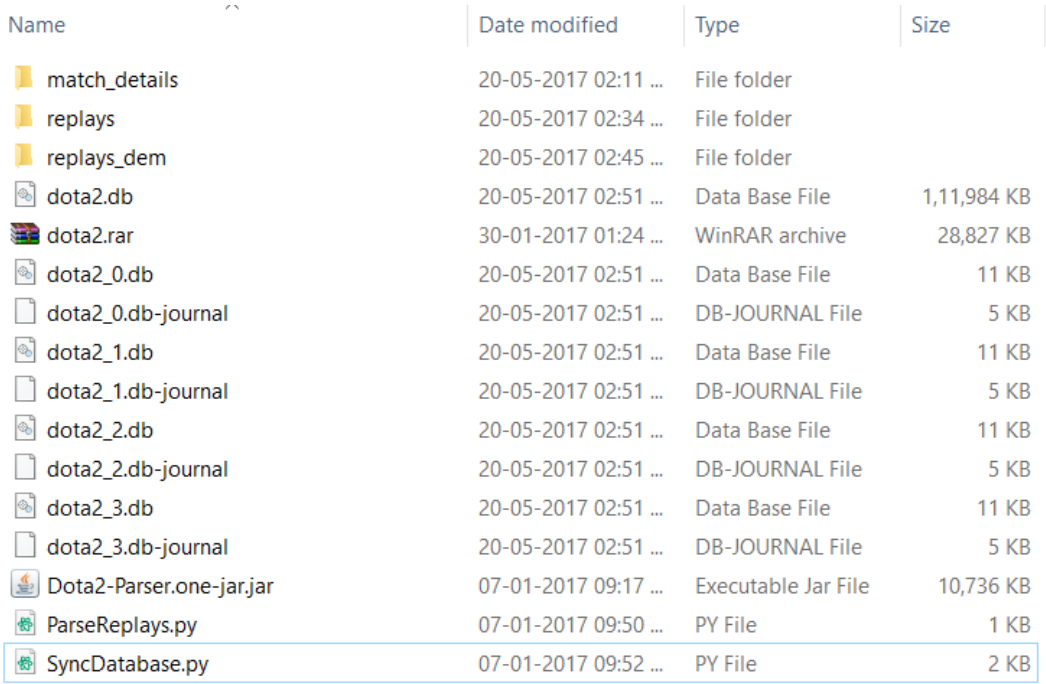
In order to parse the replay, we start from the 4500th tick and continuously check if the bounty or primary runes have spawned i.e the value associated with the primary or the bounty rune is not equal to "16777215". When this field has a value other than "16777215" the rune has spawned and this indicates that the game time is now 0th minute. The tick value is then incremented by 1800 to match the game time of 1 minute. Setting the tick is done with the help of the ControllableRunner object which is used to seek to a particular tick value.

For every game the MATCH_DATA table has one tuple which is populated once. The details of the game for every minute is inserted in the PLAYER_DATA table using a loop that increments the tick value by 1800 or 1 minute every iteration. The details are obtained using the parser to request the value of attributes like kills, deaths, assists, level, net worth etc. All of these details are extracted for every player at every minute of the game which are appended to SQLite queries which then populates the tables. After every two games have been parsed by a worker thread, a commit action is performed to ensure that in the event of a failure, all the non-committed data don't go to waste.

In the event of any exceptions, clean-up is performed which deletes all the tuples related to that game from MATCH_DATA and PLAYER_DATA tables. In addition, it also deletes the demo file, the bz2 file and the associated JSON to ensure data integrity. Clarity along with its various dependencies, classes required to populate the database are finally converted into a jar file for easy deployment. This jar file is made use of by a python script to perform everything mentioned above.

At the end of the previous step there will be 4 databases along with the main dota2.db file. The replays that were most recently parsed are present inside the 4 dota2_x.db files. These have to be synchronized with the main database file. This is done using another python script which reads all the tuples from each of the tables of each of the 4 databases and inserts them into the appropriate tables of the main database. Finally, any game that does not contain 30 tuples in the PLAYER_DATA table is deleted. The "ON DELETE CASCADE" option ensures that the appropriate game is deleted from the MATCH_DATA

table as well thereby ensuring data integrity. This database is the primary source of information for subsequent steps of the project.



Name	Date modified	Type	Size
match_details	20-05-2017 02:11 ...	File folder	
replays	20-05-2017 02:34 ...	File folder	
replays_dem	20-05-2017 02:45 ...	File folder	
dota2.db	20-05-2017 02:51 ...	Data Base File	1,11,984 KB
dota2.rar	30-01-2017 01:24 ...	WinRAR archive	28,827 KB
dota2_0.db	20-05-2017 02:51 ...	Data Base File	11 KB
dota2_0.db-journal	20-05-2017 02:51 ...	DB-JOURNAL File	5 KB
dota2_1.db	20-05-2017 02:51 ...	Data Base File	11 KB
dota2_1.db-journal	20-05-2017 02:51 ...	DB-JOURNAL File	5 KB
dota2_2.db	20-05-2017 02:51 ...	Data Base File	11 KB
dota2_2.db-journal	20-05-2017 02:51 ...	DB-JOURNAL File	5 KB
dota2_3.db	20-05-2017 02:51 ...	Data Base File	11 KB
dota2_3.db-journal	20-05-2017 02:51 ...	DB-JOURNAL File	5 KB
Dota2-Parser.one-jar.jar	07-01-2017 09:17 ...	Executable Jar File	10,736 KB
ParseReplays.py	07-01-2017 09:50 ...	PY File	1 KB
SyncDatabase.py	07-01-2017 09:52 ...	PY File	2 KB

Fig 5.7 SQLite Rollback Journals for the 4 thread level databases

SQLite implements rollback journals for atomic commits. A rollback journal is a temporary file used to implement atomic commit and rollback capabilities in SQLite. The rollback journal is always located in the same directory as the database file and has the same name as the database file except with "-journal" appended to the name. The rollback journal is created when a transaction is first started and is deleted when a transaction commits or rolls back. The rollback journal file is essential for implementing the atomic commit and rollback capabilities of SQLite. Without a rollback journal, SQLite would be unable to rollback an incomplete transaction, and a crash or power loss that occurs in the middle of a transaction corrupts the database.

5.2.3 Data Analysis

Data analysis is the process of developing answers to questions through the examination and interpretation of data. The basic steps in the analytic process consist of identifying issues, determining the availability of suitable data, deciding on which methods are appropriate for answering the questions of interest, applying the methods and

evaluating, summarizing and communicating the results. The machine learning pipeline we have used is as follows.

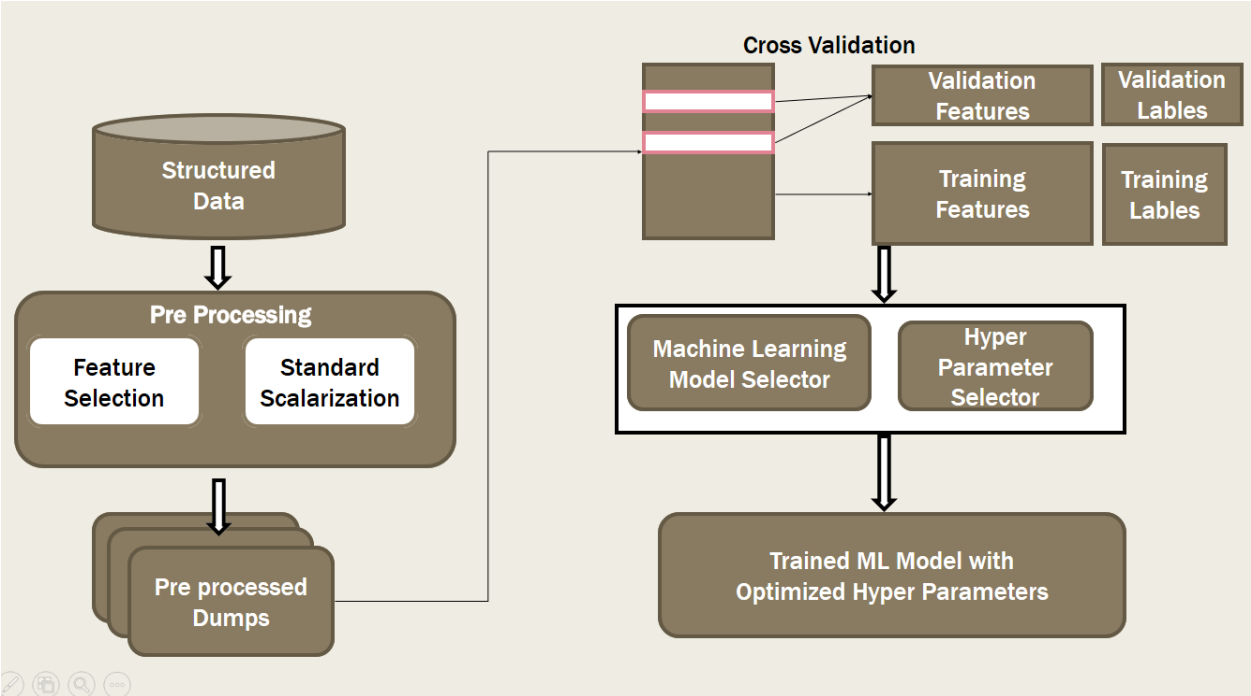


Fig 5.8 Architecture of data analysis module

This module involves building the machine learning model which will be used to perform the prediction. The data analysis is done mainly in Python using its wealth of plugins and modules. Numpy, Scipy, Scikit-Learn, Matplotlib to name a few.A feature extraction step is performed to identify the best set of features. Since this is a classification problem, more specifically a binary classification problem, a set of feasible classification algorithms was identified, such as Logistic Regression, K nearest neighbours, Random Forest Classifier and Support Vector Machine. Grid Search Cross Validation is used to identify which algorithm works best for our dataset. Hyper Parameter tuning is performed on these algorithms to obtain the best accuracy.

5.2.3.1 Data Mining

Data mining can be considered a superset of many different methods to extract insights from data. It might involve traditional statistical methods and machine learning. Data mining applies methods from many different areas to identify previously unknown

patterns from data. This can include statistical algorithms, machine learning, text analytics, time series analysis and other areas of analytics. Data mining also includes the study and practice of data storage and data manipulation.

5.2.3.2 Machine Learning

Machine learning is the subfield of computer science that, according to Arthur Samuel in 1959, gives "computers the ability to learn without being explicitly programmed." Evolved from the study of pattern recognition and computational learning theory in artificial intelligence, machine learning explores the study and construction of algorithms that can learn from and make predictions on data – such algorithms overcome following strictly static program instructions by making data-driven predictions or decisions, through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or unfeasible; example applications include email filtering, detection of network intruders or malicious insiders working towards a data breach, optical character recognition (OCR), learning to rank and computer vision.

Supervised learning algorithms are trained using labeled examples, such as an input where the desired output is known. For example, a piece of equipment could have data points labeled either “F” (failed) or “R” (runs). The learning algorithm receives a set of inputs along with the corresponding correct outputs, and the algorithm learns by comparing its actual output with correct outputs to find errors. It then modifies the model accordingly. Through methods like classification, regression, prediction and gradient boosting, supervised learning uses patterns to predict the values of the label on additional unlabeled data. Supervised learning is commonly used in applications where historical data predicts likely future events. For example, it can anticipate when credit card transactions are likely to be fraudulent or which insurance customer is likely to file a claim.

Logistic regression

Logistic regression is the appropriate regression analysis to conduct when the dependent variable is dichotomous (binary). Like all regression analyses, the logistic regression is a predictive analysis. Logistic regression is used to describe data and to

explain the relationship between one dependent binary variable and one or more nominal, ordinal, interval or ratio-level independent variables.

In logistic regression we use a different hypothesis class to try to predict the probability that a given example belongs to the "1" class versus the probability that it belongs to the "0" class. Specifically, we will try to learn a function of the form

$$P(y=1|x)= h\theta(x)=11+\exp(-\theta Tx)\equiv\sigma(\theta Tx)$$

The function $)\sigma(z)\equiv11+\exp(-z)$ is often called the "sigmoid" or "logistic" function – it is an S-shaped function that "squashes" the value of Tx into the range $[0,1][0,1]$ so that we may interpret $h\theta(x)$ as a probability.

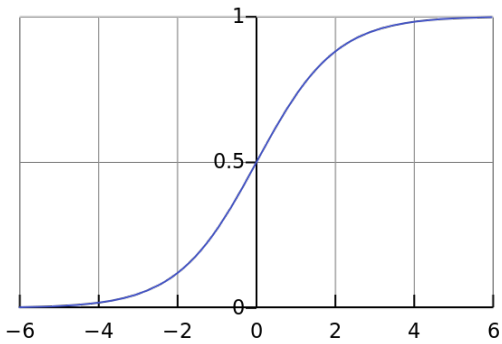


Fig 5.9 Sigmoid function

For a set of training examples with binary labels $\{(x(i),y(i)):i=1,\ldots,m\}\{(x(i),y(i)):i=1,\ldots,m\}$ the following cost function measures how well a given $h\theta h\theta$ does this:

$$J(\theta)=-\sum i(y(i)\log(h\theta(x(i)))+(1-y(i))\log(1-h\theta(x(i)))).$$

Support Vector Machine

A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (*supervised learning*), the algorithm outputs an optimal hyperplane which categorizes new examples. In which sense is the hyperplane obtained optimal? Let's consider the following simple problem: For a linearly separable set of 2D-points which belong to one of two classes, find a separating straight line.

Then, the operation of the SVM algorithm is based on finding the hyperplane that gives the largest minimum distance to the training examples. Twice, this distance receives the important name of margin within SVM's theory. Therefore, the optimal separating hyperplane *maximizes* the margin of the training data.

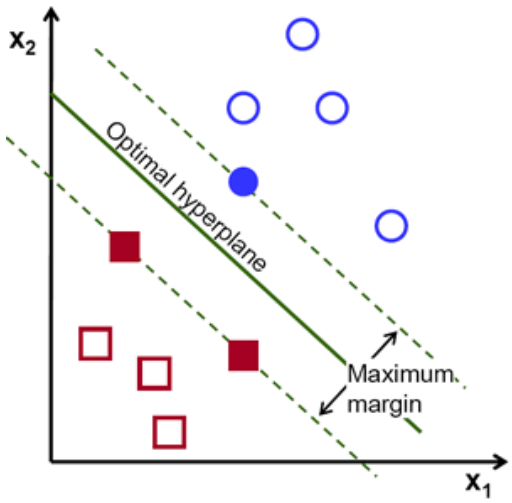


Fig 5.10 SVM classification

Random forest

The random forest is an ensemble approach that can also be thought of as a form of nearest neighbor predictor. Ensembles are a divide-and-conquer approach used to improve performance. The main principle behind ensemble methods is that a group of "weak learners" can come together to form a "strong learner". Each classifier, individually, is a "weak learner," while all the classifiers taken together are a "strong learner". The random forest starts with a standard machine learning technique called a "decision tree" which, in ensemble terms, corresponds to our weak learner. In a decision tree, an input is entered at the top and as it traverses down the tree the data gets bucketed into smaller and smaller sets

k-nearest neighbors

k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression.

In k-NN classification, the output is a class membership. An object is classified by a majority vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor. In k-NN regression, the output is the property value for the object. This value is the average of the values of its k nearest neighbors.

Feature engineering

Feature engineering or feature selection is an import part in any machine learning pipeline. Our original feature set consisted of 170 features. This had to be reduced to an acceptable amount. There are two ways this could be done. By using statistical methods like Principal Component Analysis which is a dimensionality reduction algorithm. The other approach is to apply domain knowledge and engineer the features.

By modifying the features and transforming them we have reduced the number of features to 18. This makes the training a lot faster and the accuracy also acceptable. The techniques used is described as follows.

The data set consists of around 8052 games. Each of these are defined by 170 features. Each hero contributes 17 features and since Dota 2 consists of 10 heroes we have a 170 feature sample. The initial approach was as follows, we rearranged the 1x170 into a 10x17 feature set, now each row represents the features of one hero. Then we add the first 5 rows which consists of hero 1 to hero 5's features, then we add the last 5 rows which is hero 6 to hero 10 features. We now end up with a 2x17 feature set. This is flattened to form a 1x34 feature sample which represents the attributes at a team level as opposed to the previous hero level attributes. This is how we reduce 170 features to 34 features for each match. Reducing features is crucial since we can avoid the "curse of dimensionality" that is we can effectively reduce the amount of training data needed by reducing the number of features and our dataset dosen't become sparse.

The next approach was to filter the number of attributes based on a feature ranking step. The most optimal features were identified and a set of 9 features per hero were identified.

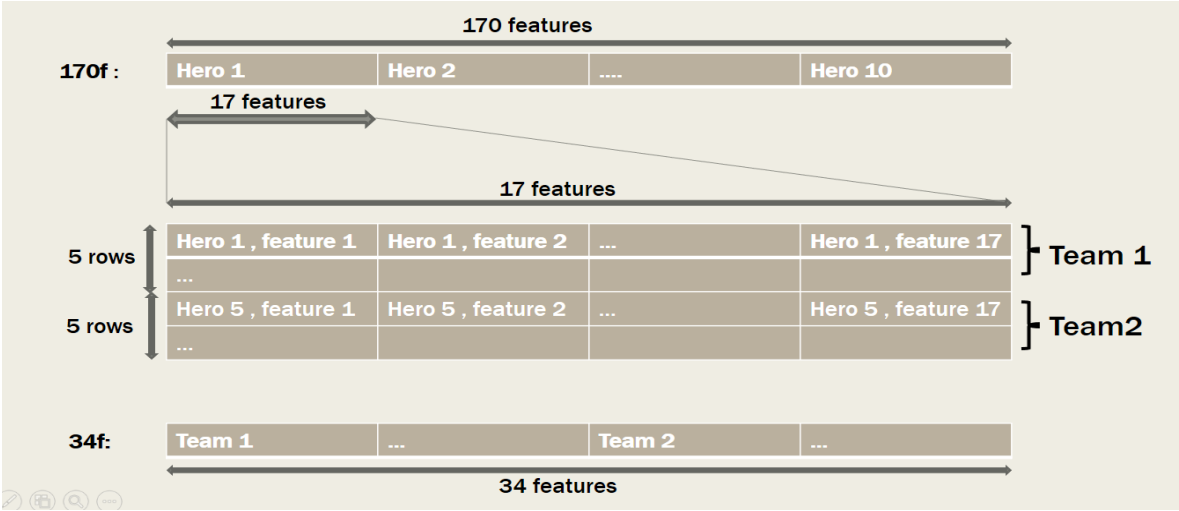


Fig 5.11 Feature engineering

These features are listed below

- HERO_ASSISTS
- HERO_DEATHS
- HERO_DENY_COUNT
- HERO_KILLS
- HERO_LAST_HIT_COUNT
- HERO_NET_WORTH
- HERO_TOTAL_GOLD_EARNED
- HERO_TOTAL_XP_EARNED
- HERO_TOWER_KILLS

Since each hero contributes 9 features, the compete feature vector consists of 10x9 or a 1x90 feature vector. A similar process as described above was used to reduce this 1x90 feature vector to 1x18, which gives the most optimal features.

Predictor

Once the model is trained, it is used to predict ongoing Dota 2 games. A model for each minute of the game is trained. A list of live games are obtained, To choose an appropriate model for the prediction, the current duration of the game is rounded off to the next minute, As an example is the game has progressed 14 minutes and 40 seconds, the prediction model trained using the 15^{th} minute data set is used. A snapshot of the predictor in action is shown in the figure below.

```
Last login: Thu May 18 17:33:24 on ttys000
DWMOHAN-M-C2HZ:predictor dwmohan$ python league_predictor.py
Predicting matchid: 3191456060   Duration: 39 min
Dire has a  73.92% probablity of winning
Model accuracy at  30  min is: 97.64%

Predicting matchid: 3191496457   Duration: 14 min
Radiant has a  50.34% probablity of winning
Model accuracy at  14  min is: 78.80%

Predicting matchid: 3191508285   Duration: 11 min
Radiant has a  56.09% probablity of winning
Model accuracy at  11  min is: 74.36%

Predicting matchid: 3191511120   Duration: 8 min
Dire has a  57.15% probablity of winning
Model accuracy at  8  min is: 70.80%

Predicting matchid: 3191497459   Duration: 15 min
Radiant has a  64.26% probablity of winning
Model accuracy at  15  min is: 80.25%
```

Fig 5.12 Output of predictor

5.2.4 Frontend (Website)

The website is built using python and Django as a framework. It uses our trained model to predict the outcome of live Dota 2 league games. The data needed to predict the outcome of these games is acquired using a Steam API. The website first displays the details of currently ongoing games and when clicked on a particular game, it displays the prediction in terms of win probability of each team via a pie chart. Apart from the prediction it also displays various other details of each of the 10 players in the game via a player statistics table.

Django is a free and open source web framework written in python. It uses the Model View Template (MVT) architectural pattern. It focuses on reusability and rapid development. It has an active community and great documentation. It is maintained by the Django Software Foundation, an independent organization established as a 501 non-profit.

5.2.4.1 Basic structure of website app

Each app in our website has the following parts:

- URLS are used to direct the user to specific resources such as HTML pages, pictures, videos etc.
- VIEW represents what is displayed to the user.
- TEMPLATE represents structure of a file such as HTML page.
- MODEL provides mechanisms to manage and query records in a database.

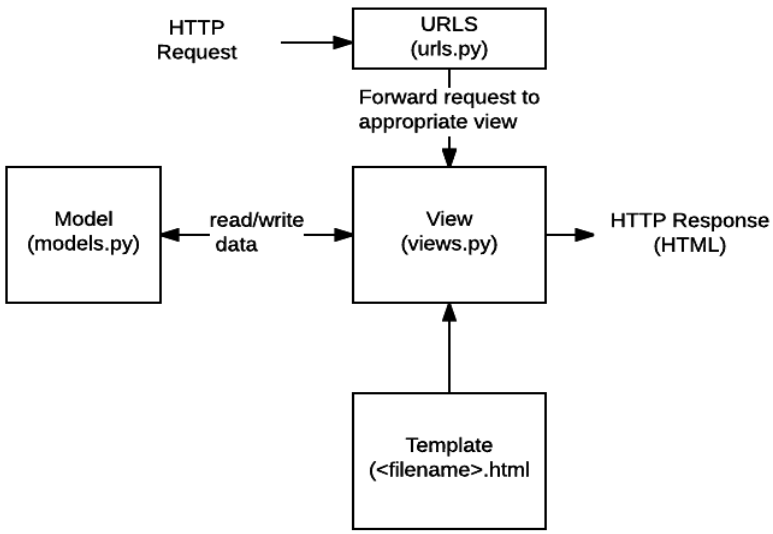


Fig 5.13 Basic Django code

5.2.4.2 Apps Used in Frontend

As per Django, a website is a collection of apps. For our project we create 2 apps ie, livegames and prediction.

livegames app

livegames uses a steam API to get details of current Dota2 league games that are being played. The API needs a developer key which must be procured by making a prime account in steam. We then iterate over all the matches and display details of only certain suitable matches. Match details are displayed one after another in rows.

Each row has:

- matchid,
- score of radiant and dire
- time elapsed
- no. of viewers watching
- details of dire and radiant

Every match being played has an ID so that it can be uniquely stored and accessed from the steam servers. This is called “matchid”. Each of the two teams are given an overall “score” based on the performance of all 5 players in each team. Things like net gold, net XP, kills, deaths, assists etc of each player is taken into account to generate this score. Although overall score may indicate that the team with higher score may win, but that is not always the case since Dota 2 is a very dynamic game and a particular team may be studying the opponent team and hence isn’t on the offensive. As soon as they see their chance they go on offense and increase their score. The “time elapsed” specifies how much time from the beginning of the match has elapsed. There is functionality to watch ongoing matches that Steam provides. “Number of viewers” specifies how many people are currently watching a particular match. This value is a good indicator that a particular match is interesting and both teams are at their best.

The left side displays details of radiant and right side displays details of dire. It also displays the score of each team along with icons of heroes that the players are currently playing as. The website is dynamic in the sense that it constantly displays updated details. This is done with the help of a timer located at the bottom of the page. As soon as the timer reaches 0, the API is queried again and the updated details are displayed instead of the old one.

Sequence of steps:

- Get details of current Dota2 league games via steam API: https://api.steampowered.com/IDOTA2Match_570/GetLiveLeagueGames/v0001/?key=ACABEB1FD8894A44B2A5AB4B79209C75
- Iterate over all matches and display details of matches which are not corrupted, have already started and have 5 players each in both the teams.
- At the end of every 20 seconds refresh the page with updated details by querying.

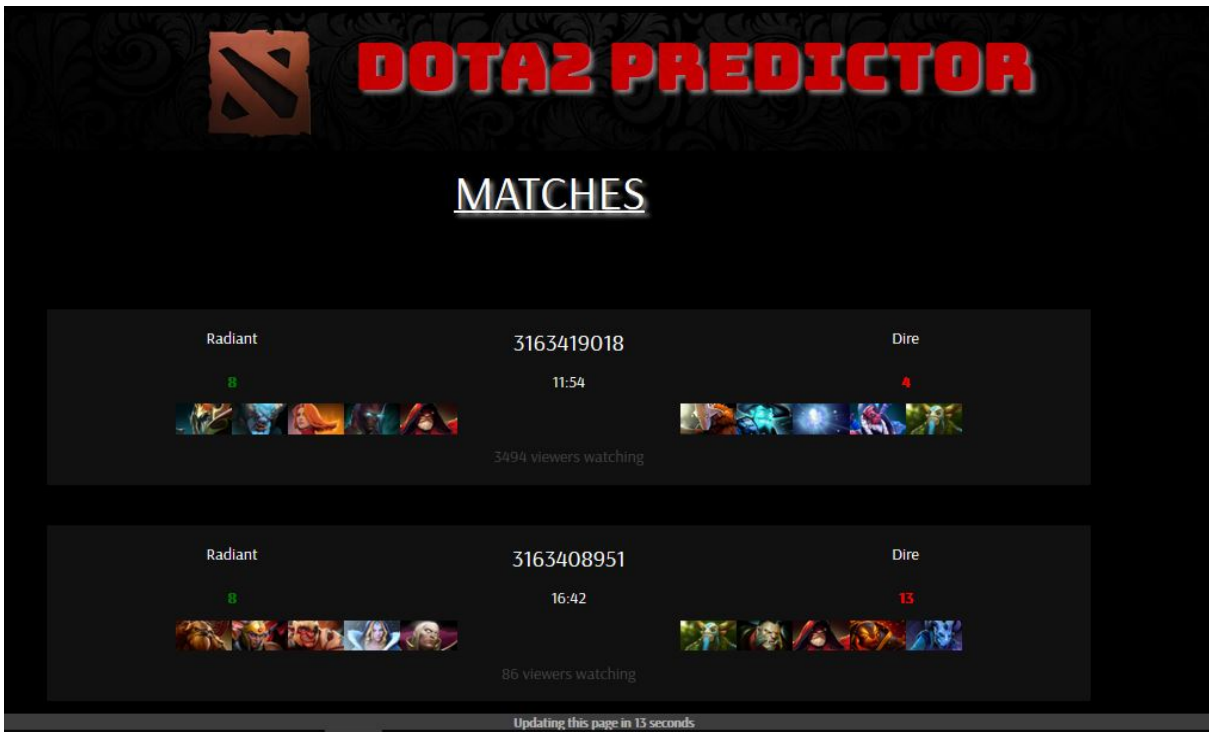


Fig 5.14 Snapshot of livegames app

prediction app

Once we click on a particular match, the prediction details are displayed using the prediction app which uses our trained model as a backend.

For every match, the details displayed are:

- matchid
- match duration
- net worth of radiant and dire
- predicted winner (either radiant or dire)
- win probability of radiant and dire
- prediction accuracy
- a pie chart displaying win probability of radiant and dire graphically
- player statistics in tabular format
- score of radiant and dire
- minimap showing the status of towers of both the teams
- a bar chart displaying net worth of each player and teams graphically

The minimap as the name suggests is a scaled down version of the actual Dota 2 map. It shows the status of the towers (either standing or destroyed) on the map. This is updated in real time that is, if a particular tower is destoyed then it is no longer shown on the minimap. This allows us to determine which team is dominating which lane.

The player statistics table gives the details of all 10 players in the game which is helpful in summarizing which player is contributing the most vs least in the team.The bar chart has a button which when clicked switches view from team to individual players net worth. Just like overall score, “net worth” of a player is calculated by Steam based on his in game performance, values of attributes like net gold, net XP, Kills/Deaths/Assists etc. This is again a good way to visualize which player is most effective in the team.The bar chart and the pie chart are made using Google Charts API which is an interactive web service that creates graphical charts from user supplied data. It’s a very good way to visualize our data and also allows us to customize it to fit our website’s look and feel. . Google charts is loaded via the url: https://www.gstatic.com/charts/loader.js in the <script> tag.

The player statistics table displays the following information:

1. Name of the player
2. Level of the player
3. Net worth
4. Gold earned by the player
5. Number of times the player died, killed opponent heroes, assisted a team member in killing an opponent hero
6. Last hits and denies

IMPLEMENTATION

```
/*Downloading the JSONs of relevant replays*/
import subprocess
import urllib2
import json

def get_match_details(match_id):
        res =urllib2.urlopen ("https://api.steampowered.com/IDOTA2Match_570/
            /GetMatchDetails/V001/?match_id="+str(match_id)+ "&key=ACAB
            EB1FD8894A44B2A5AB4B79209C75")
        string = res.read().decode('utf-8')
        json_obj = json.loads(string)
        return json_obj

NUM_REPLAYS_TO_DOWNLOAD = 3
CONFIG_FILE = "./config/dwaraka0071_config"

match_ids = []
duration_list = []
num=1
previous_matchid = ""

while num<= NUM_REPLAYS_TO_DOWNLOAD:
        if len(previous_matchid)==0
```

```
res = urllib2.urlopen ("https://api.steampowered.com/IDOTA2Match_570/
        GetMatchHistory/V001/?format=json&skill=3&key=ACABEB1FD
        8894A44B2A5AB4B79209C75")
string = res.read().decode('utf-8')
json_obj = json.loads(string)
matches = json_obj['result']['matches']
previous_matchid = str(matches[99]['match_id'])
continue
else:
res = urllib2.urlopen ("https://api.steampowered.com/IDOTA2Match_570/
        GetMatchHistory/V001/?start_at_match_id="+previous_matchid+
        "&skill=3&key=ACABEB1FD8894A44B2A5AB4B79209C75")
string = res.read().decode('utf-8')
json_obj = json.loads(string)
matches = json_obj['result']['matches']
if len(matches)!=100:
        break
previous_matchid = str(matches[99]['match_id'])

for i in range(0,100):
        if num<=NUM_REPLAYS_TO_DOWNLOAD:
                match_id = matches[i]['match_id']
                duration = get_match_details(match_id)['result']['duration']
```

```
                    if duration > 1800:
                        match_ids.append(matches[i]['match_id'])
                        duration_list.append(duration)
                        print num,"Selected",
                        print "Match_id:",match_id,
                        print "Duration:",duration," seconds"
                        num+=1


argv = CONFIG_FILE
for match_id in match_ids:
        argv+=" "+str(match_id)


print "Calling dota client"
command = "node match_details.js "+argv
subprocess.call(command, shell=True)
print "Finished downloading"
**************************************************************
/*Code to download the replay files*/
import threading
import os
import urllib2
import json
import sys
import time
```

```
import requests


NUM_THREADS = 4

BROKEN_THRESHOLD = 15


class myThread (threading.Thread):

        def __init__(self, threadID,urlList):

                threading.Thread.__init__(self)

                self.threadID = str(threadID)

                self.urlList = urlList

        def run(self):

                print "Starting Thread "+self.threadID+"\n",

                for url in self.urlList:

                        print "[Thread "+self.threadID+"]"+"Downloading "+url+"\n",

                        downloadFile(url,self.threadID):

                print "Exiting Thread "+self.threadID+"\n",


        def downloadFile(url,threadID):

                localFilename = url.split('/')[-1]

                with open("../../resources/replays"+ '/' + localFilename, 'wb') as f:

                start = time.clock()

                r = requests.get(url, stream=True)

                total_length = int(r.headers.get('content-length'))

                if total_length==0:
```

```
                print "[Thread "+threadID+"]"+" Time Out!"+"\n",
                return
            dl = 0
            if total_length is None:
            f.write(r.content)
            else:
                for chunk in r.iter_content(1024):
                    dl += len(chunk)
                    f.write(chunk)
                    done = int(50 * dl / total_length)
                        if (time.clock()-start) > 10:
                          if ((dl//(time.clock() - start))/(1024)) < BROKEN_
                                                                THRESHOLD:
                            print "[Thread "+threadID+"]"+"Broken URL"+"\n",
                            return


files =  os.listdir("../../resources/match_details")
urls = []
for f in files:
        if "DS_Store" in f:
            continue
        match_id = f.split(".")[0]
        json_obj = json.load(open("../../resources/match_details/"+f,"r"))
        replay_salt =  json_obj['match']['replay_salt']
```

```
            cluster = json_obj['match']['cluster']
            if os.path.exists("../../resources/replays/"+match_id+"_"+str(replay_salt) +
              ".dem.bz2")==False:
                url = "http://replay"+str(cluster)+".valve.net/570/"+match_id+"_"+
                    "_"+str(replay_salt)+".dem.bz2";
                urls.append(url)
                urls.append(url)



urlList = [urls[i::NUM_THREADS] for i in range(NUM_THREADS)]


threads = []
for i in range(0,NUM_THREADS):
            thread = myThread(i,urlList[i])
            thread.start()
            threads.append(thread)


for t in threads:
            t.join()


print "Exiting Main Thread"
**********************************************************************
/*Code to extract the compressed files*/
import threading
```

```
import os
import bz2
import sys


NUM_THREADS = 4


class myThread (threading.Thread):
        def __init__(self, threadID,bz2List):
            threading.Thread.__init__(self)
            self.threadID = str(threadID)
            self.bz2List = bz2List
        def run(self):
            print "Starting Thread "+self.threadID+"\n",
            for bz2file in self.bz2List:
                print "[Thread "+self.threadID+"]"+"Extracting "+bz2file+"\n",
                extract(self,bz2file)
           print "Exiting Thread "+self.threadID+"\n",


def extract(self,bz2file):
        match_id = bz2file.split(".")[0]
        zipfile = bz2.BZ2File("../../resources/replays/"+bz2file)
        try:
            data = zipfile.read()
            open("../../resources/replays_dem/"+match_id+".dem", 'wb').write(data)
```

```
                except EOFError:
                    zipfile.close()
                    print "[Thread "+self.threadID+"]"+"EOF error:
                        deleted"+str(match_id)+"\n",

                    #os.remove("../../resources/replays/"+bz2file)
                    os.remove("../../resources/match_details/"+match_id.split("_")[0]+".json")
                except IOError:
                    zipfile.close()
                    print "[Thread "+self.threadID+"]"+"IO error: deleted"+
                        str(match_id)+"\n",
                    #os.remove("../../resources/replays/"+bz2file)
                    os.remove("../../resources/match_details/"+match_id.split("_")[0]+".json")

files = os.listdir("../../resources/replays")
bz2files = []

for f in files:
                if "DS_Store" in f:
                    continue
                match_id = f.split(".")[0]
                if os.path.exists("../../resources/replays_dem/"+match_id+".dem") == False:
                    bz2files.append(f)
```

```
bz2List = [bz2files[i::NUM_THREADS] for i in range(NUM_THREADS)]


threads = []
for i in range(0,NUM_THREADS):
        thread = myThread(i,bz2List[i])
        thread.start()
        threads.append(thread)


for t in threads:
        t.join()


print "Exiting Main Thread"
**********************************************************************
/* Data Extraction code*/
package dwaraka.ddns.net;


import java.io.File;
import java.util.ArrayList;


public class Main {
        public static void main(String[] args) throws Exception {
            File folder = new File("replays_dem");
            File[] listOfFiles = folder.listFiles();
           ArrayList<String> list0=new ArrayList<String>();
```

```
ArrayList<String> list1=new ArrayList<String>();
 ArrayList<String> list2=new ArrayList<String>();
ArrayList<String> list3=new ArrayList<String>();
int count = 0;
for(File f:listOfFiles){
        switch(count){
                        case 0:
                                list0.add(f.getName());
                                break;
                         case 1:
                                list1.add(f.getName());
                                break;
                        case 2:
                                ist2.add(f.getName());
                                break;
                        case 3:
                                list3.add(f.getName());
                                break;
                        }
                count = (count+1)%4;
                        }
PlayerDatabase playerDatabase = new PlayerDatabase("dota2.db");
ArrayList<Long> matchIds = playerDatabase.getMatchIds();
System.out.println("Number of matchIds found = "+matchIds.size());
```

```
playerDatabase.closeConnection();
WorkerThread worker0 = new WorkerThread("0",list0,matchIds);
WorkerThread worker1 = new WorkerThread("1",list1,matchIds);
WorkerThread worker2 = new WorkerThread("2",list2,matchIds);
WorkerThread worker3 = new WorkerThread("3",list3,matchIds);
final long startTime = System.currentTimeMillis();
try{
                worker0.t.join();
                worker1.t.join();
                worker2.t.join();
               worker3.t.join();
}catch(InterruptedException e){
                System.out.println("Main interrupted");
                                        }
               final long endTime = System.currentTimeMillis();
              System.out.println("Total execution time: " + (endTime - startTime) );
                                }
                  }
```

```
package dwaraka.ddns.net;

import java.sql.*;
import java.io.File;
import java.util.ArrayList;
```

```
class PlayerDatabase{
        Connection c;
        String databaseName;

        PlayerDatabase(String databaseName){
            this.databaseName = databaseName;
            try {
                    File file = new File(databaseName);
                    Class.forName("org.sqlite.JDBC");
            System.out.println("Checking for database file "+databaseName+"...");
            if(file.exists())
            c = DriverManager.getConnection("jdbc:sqlite:"+databaseName);
            else{
            c = DriverManager.getConnection("jdbc:sqlite:"+databaseName);
            createDatabase();
                }
        c.setAutoCommit(false);
        Statement stmt = c.createStatement();
        stmt.executeUpdate("delete from match_data where match_id in (select
match_id from player_data group by match_id having count(*)<30);");
        stmt.executeUpdate("delete from player_data where match_id in (select
match_id from player_data group by match_id having count(*)<30);");
        stmt.close();
        c.commit();
```

```
                } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
                }
           System.out.println("Opened database "+databaseName+" successfully");
                                }

 public void closeConnection(){
        try {
                c.close();
            } catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
                                }
           System.out.println("Database "+databaseName+" close successfully");
                                }

  public void createDatabase(){
         Statement stmt = null;
         try {
                stmt = c.createStatement();
                String sqlMatchData = "CREATE TABLE MATCH_DATA"+
" GAME_MODE INT," +" DURATION INT," +" MATCH_OUTCOME TEXT," +
" PLAYER_0_HERO_ID INT," +" PLAYER_1_HERO_ID INT," +
```

```
" PLAYER_2_HERO_ID INT," +" PLAYER_3_HERO_ID INT," +
" PLAYER_4_HERO_ID INT," +" PLAYER_5_HERO_ID INT," +
" PLAYER_6_HERO_ID INT," +" PLAYER_7_HERO_ID INT," +
" PLAYER_8_HERO_ID INT," +" PLAYER_9_HERO_ID INT)";
                stmt.executeUpdate(sqlMatchData);
                System.out.println("Table MATCH_DATA created successfully");
                String sqlPlayerData = "CREATE TABLE PLAYER_DATA " +
                                                "(MATCH_ID INT," +" TIME_ELAPSED INT,"
+getCreateStatement() +" PRIMARY KEY (MATCH_ID,TIME_ELAPSED)," +"
FOREIGN KEY (MATCH_ID) REFERENCES MATCH_DATA(MATCH_ID) ON
DELETE CASCADE)";
                stmt.executeUpdate(sqlPlayerData);
                System.out.println("Table PLAYER_DATA created successfully");
                stmt.close();
                        } catch ( Exception e ) {
                            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
                            System.exit(0);
                                                }
                                    }
public void insertPlayerData(String match_id,int time,String rawStatement){
                Statement stmt = null;
                try{
                            stmt = c.createStatement();
                            String sql = "INSERT INTO PLAYER_DATA VALUES("
+ match_id +","+Integer.toString(time)+",+ rawStatement + ");";
```

```
                    stmt.executeUpdate(sql);
                    stmt.close();
                    //c.commit();
                        }catch ( Exception e ) {
                    System.err.println( e.getClass().getName() + ": " + e.getMessage() );
                    System.exit(0);
                                                }
                                    }

public void insertMatchData(String rawStatement){
            Statement stmt = null;
                    try{
                            c.setAutoCommit(false);
                            stmt = c.createStatement();
                            String sql = "INSERT INTO MATCH_DATA VALUES("
                                        + rawStatement + ");";
                            stmt.executeUpdate(sql);
                            stmt.close();
                            //c.commit();
                        }catch ( Exception e ) {
                System.err.println( e.getClass().getName() + ": " + e.getMessage() );
                System.exit(0);
                                    }
                }
```

```
 public void performCommit(){
        try{
            c.commit();
          }catch ( Exception e ) {
              System.err.println( e.getClass().getName() + ": " + e.getMessage() );
              System.exit(0);
                                }
   }
public boolean exists(String match_id){
              boolean rval = true;
              Statement stmt = null;
              try {
                    stmt = c.createStatement();
ResultSet rs = stmt.executeQuery( "SELECT COUNT(*) AS NUM_ROWS FROM
MATCH_DATA WHERE MATCH_ID = "+match_id+";" );
                    if(rs.getInt("NUM_ROWS")==0)
                              rval = false;
                              rs.close();
                              stmt.close();
                  } catch ( Exception e ) {
              System.err.println( e.getClass().getName() + ": " + e.getMessage() );
              System.exit(0);
                                }
              return rval;
```

```
                }
public ArrayList<Long> getMatchIds(){
        ArrayList<Long> matchIds = new ArrayList<Long>();
            try{
                Statement stmt = c.createStatement();
                ResultSet rs = stmt.executeQuery( "SELECT * FROM
                                                    MATCH_DATA")
                while(rs.next())
                    matchIds.add(rs.getLong("MATCH_ID"));
                stmt.close();
             }catch ( Exception e ) {
            System.err.println( e.getClass().getName() + ": " + e.getMessage() );
            System.exit(0);
                                }
            return matchIds;
                        }

private String getCreateStatement(){
        String createStatement = "";
        for(int i=0;i<10;i++){
                String partialCreateStatement =
    " PLAYER_"+Integer.toString(i)+"_ASSISTS INT," +
    " PLAYER_"+Integer.toString(i)+"_CAMPS_STACKED INT," +
    " PLAYER_"+Integer.toString(i)+"_DEATHS INT," +
```

```
            " PLAYER_"+Integer.toString(i)+"_DENY_COUNT INT," +
            " PLAYER_"+Integer.toString(i)+"_HEALING REAL," +
            " PLAYER_"+Integer.toString(i)+"_KILLS INT," +
            " PLAYER_"+Integer.toString(i)+"_LAST_HIT_COUNT INT," +
            " PLAYER_"+Integer.toString(i)+"_LEVEL INT," +
            " PLAYER_"+Integer.toString(i)+"_NET_WORTH INT," +
            " PLAYER_"+Integer.toString(i)+"_OBSERVER_WARDS_PLACED INT," +
            " PLAYER_"+Integer.toString(i)+"_ROSHAN_KILLS INT," +
            " PLAYER_"+Integer.toString(i)+"_RUNE_PICKUPS INT," +
            " PLAYER_"+Integer.toString(i)+"_SENTRY_WARDS_PLACED INT," +
            " PLAYER_"+Integer.toString(i)+"_STUN_DURATION REAL," +
            " PLAYER_"+Integer.toString(i)+"_TOTAL_GOLD_EARNED INT," +
            " PLAYER_"+Integer.toString(i)+"_TOTAL_XP_EARNED INT," +
            " PLAYER_"+Integer.toString(i)+"_TOWER_KILLS INT," ;
            createStatement+=partialCreateStatement;
                                        }
  return createStatement;
      }
}
**********************************************************************
package dwaraka.ddns.net;


import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import skadistats.clarity.decoder.Util;
```

```
import skadistats.clarity.model.EngineType;

import skadistats.clarity.model.Entity;

import skadistats.clarity.model.FieldPath;

import skadistats.clarity.processor.entities.Entities;

import skadistats.clarity.processor.entities.UsesEntities;

import skadistats.clarity.processor.runner.ControllableRunner;

import skadistats.clarity.source.MappedFileSource;

import skadistats.clarity.util.TextTable;


import org.json.simple.JSONArray;

import org.json.simple.JSONObject;

import org.json.simple.parser.JSONParser;


import java.io.IOException;

import java.io.FileReader;

import java.io.File;

import java.io.BufferedReader;

import java.io.InputStream;

import java.io.InputStreamReader;


@UsesEntities

class Replay{

        private final ControllableRunner runner;
```

```
public Replay(String fileName) throws IOException,InterruptedException {
                runner=newControllableRunner(new
                        MappedFileSource(fileName)).runWith(this);
                runner.seek(2);
  }

public void seek(int tick) throws InterruptedException{
                runner.seek(tick);
  }

public void stop(){
                runner.halt();
               }

public int getLastTick() throws IOException{
                return runner.getLastTick();
               }

public int get HeroId (intplayer_pos){
returngetIntResource("CDOTA_PlayerResource","m_vecPlayerTeamData.000"+
player_pos+".m_nSelectedHeroID");
                                }
publicintgetKills(intplayer_pos){
return getIntResource("CDOTA_PlayerResource","m_vecPlayerTeamData.000"+
player_pos+".m_iKills");
```

```
                                        }
public int getAssists (intplayer_pos){
return getIntResource("CDOTA_PlayerResource","m_vecPlayerTeamData.000"+
player_pos+".m_iAssists");
                                        }
public int getDeaths (intplayer_pos){
return getIntResource("CDOTA_PlayerResource","m_vecPlayerTeamData.000"+
player_pos+".m_iDeaths");
                                        }
public int getLevel (intplayer_pos){
return getIntResource("CDOTA_PlayerResource","m_vecPlayerTeamData.000"+
player_pos+".m_iLevel");
                                        }

public int getNetWorth (intplayer_pos){
return getIntResource("CDOTA_DataSpectator","m_iNetWorth.000"+
player_pos);
                                        }
public int getTotalGoldEarned(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iTotalEarnedGold");
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iTotalEarnedGold");
```

```
                                        }
public int getTotalXPEarned(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iTotalEarnedXP");
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iTotalEarnedXP");
                                        }
public int getDenyCount(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iDenyCount");
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iDenyCount");
                                        }
public int getLastHitCount(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iLastHitCount");

return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iLastHitCount");
                                        }
public float getStunDuration(int player_pos){
```

```
                if(player_pos<5)
return getFloatResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_fStuns");
return getFloatResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_fStuns");
                                        }
public float getHealing(int player_pos){
                if(player_pos<5)
return getFloatResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_fHealing");
return getFloatResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_fHealing");
                                        }
public int getTowerKills(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iTowerKills");
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iTowerKills");
                                        }
public int getRoshanKills(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iRoshanKills");
```

```
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iRoshanKills");
                                        }
public int getObserverWardsPlaced(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iObserverWardsPlaced");
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iObserverWardsPlaced");
                                        }
public int getSentryWardsPlaced(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iSentryWardsPlaced");
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iSentryWardsPlaced");
                                        }
public int getRunePickups(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iRunePickups");
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iRunePickups");
                                        }
```

```
public int getCampsStacked(int player_pos){
                if(player_pos<5)
return getIntResource("CDOTA_DataRadiant","m_vecDataTeam.000"+
player_pos+".m_iCampsStacked");
return getIntResource("CDOTA_DataDire","m_vecDataTeam.000"+
(player_pos-5)+".m_iCampsStacked");
                                        }
public int getPrimaryRune(){
return getIntResource("CDOTA_DataSpectator","m_hPrimaryRune");
                                        }

public int getBountyRune_1(){
return getIntResource("CDOTA_DataSpectator","m_hBountyRune_1");
                                        }

private int getIntResource(String entityName,String fieldName){
                Entity entity = getEntity(entityName);
                FieldPath fieldPath = entity.getDtClass().getFieldPathForName(fieldName);
                return entity.getPropertyForFieldPath(fieldPath);
                                        }

public float getFloatResource(String entityName,String fieldName){
                Entity entity = getEntity(entityName);
                FieldPath fieldPath = entity.getDtClass().getFieldPathForName(fieldName);
```

```
        return entity.getPropertyForFieldPath(fieldPath);
                                }

private Entity getEntity(String entityName) {
return runner.getContext().getProcessor(Entities.class).getByDtName(entityName);
                                }

public String getPlayerData() throws Exception{
        String playerData = "";
        for(int i=0;i<9;i++){
            String individualPlayerData = getIndividualPlayerData(i);
            playerData+=individualPlayerData+",";
                                }
        playerData+=getIndividualPlayerData(9);
        return playerData;
                                }

private String getIndividualPlayerData(int player_id) throws Exception{
        String individualPlayerData="";
        individualPlayerData+=Integer.toString(getAssists(player_id))+",";
        individualPlayerData+=Integer.toString(getCampsStacked(player_id))+",";
        individualPlayerData+=Integer.toString(getDeaths(player_id))+",";
        individualPlayerData+=Integer.toString(getDenyCount(player_id))+",";
        individualPlayerData+=Float.toString(getHealing(player_id))+",";
```

```
            individualPlayerData+=Integer.toString(getKills(player_id))+",";
            individualPlayerData+=Integer.toString(getLastHitCount(player_id))+",";
            individualPlayerData+=Integer.toString(getLevel(player_id))+",";
            individualPlayerData+=Integer.toString(getNetWorth(player_id))+",";
            individualPlayerData+=Integer.toString(getObserver(player_id))+",";
            individualPlayerData+=Integer.toString(getRoshanKills(player_id))+",";
            individualPlayerData+=Integer.toString(getRunePickups(player_id))+",";
            individualPlayerData+=Integer.toString(getSentryPlaced(player_id))+",";
            individualPlayerData+=Float.toString(getStunDuration(player_id))+",";
            individualPlayerData+=Integer.toString(getTotalGold(player_id))+",";
            individualPlayerData+=Integer.toString(getTotalXPEarn(player_id))+",";
            individualPlayerData+=Integer.toString(getTowerKills(player_id));
            return individualPlayerData;
                                    }
}
**********************************************************************
package dwaraka.ddns.net;

import java.io.File;
import java.io.FileReader;
import java.util.ArrayList;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
```

```
import org.json.simple.parser.JSONParser;


class WorkerThread implements Runnable{
                Thread t;
                PlayerDatabase playerDatabase;
                ArrayList<String> listOfFiles;
                ArrayList<Long> matchIds;
                String threadID;
                int num;
                int max;


        WorkerThread(StringthreadID,ArrayList<String>listOfFiles,ArrayList<Long>
matchIds){
                t = new Thread(this,threadID);
                this.listOfFiles = listOfFiles;
                this.threadID = threadID;
                this.matchIds = matchIds;
                max = listOfFiles.size();
                num = 0;
                t.start();
            }


        public void run(){
                try{
                    playerDatabase = new PlayerDatabase("dota2_"+threadID+".db");
```

```
for(String s:listOfFiles){
File f = new File("replays_dem/"+s);
try{
num++;
processMatch(f.getName());
if((num%2)==0)
playerDatabase.performCommit();
}
catch(Exception e){
cleanUp(f.getName());
}
}
playerDatabase.performCommit();
playerDatabase.closeConnection();
}
catch(Exception e){

}
}

private boolean exists(String match_id){
Long matchId = Long.parseLong(match_id);
for(Long i:matchIds)
{
```

```
                    if(i.longValue()==matchId.longValue())
                        {
                          return true;
                        }
                }
                    return false;
            }

private void processMatch(String fName) throws Exception{
                String match_id = fName.split("[.]")[0].split("_")[0];
                if(!exists(match_id))
                    {
                System.out.println("[Thread"+threadID+"]"+"["+(max-
num)+"]"+":Processing match "+match_id+"...");
                Replay replay = new Replay("replays_dem/"+fName);
                int first_tick = 4500;
                if(Long.parseLong(match_id)>=2842495294L)
                while(replay.getBountyRune_1()==16777215)replay.seek(first_tick++);
               else
               while(replay.getPrimaryRune()==16777215)replay.seek(first_tick++);
               int last_tick = 30*1800+first_tick;
               first_tick = first_tick+1800;
               replay.seek(first_tick);
               insertMatchDetails(match_id,replay);
               int current_minute = 0;
```

```
                for(int t=first_tick;t<=last_tick;t+=1800)
                        {
                                replay.seek(t);
                                current_minute++;
playerDatabase.insertPlayerData(match_id,current_minute,replay.getPlayerData());
                         }
                replay.stop();
                   }
              }

private void cleanUp(String fName) throws Exception{
                String match_id_replay_salt = fName.split("[.]")[0];
                new File("replays_dem/"+match_id_replay_salt+".dem").delete();
                new File("replays/"+match_id_replay_salt+".dem.bz2").delete();
               new File("match_details/"+match_salt.split("_")[0]+".json").delete();
               System.out.println("[Thread"+threadID+"]"+":Deletedcorruptreplay
"+match_id_replay_salt);
                                        }

private void insertMatchDetails(String match_id,Replay replay)throws Exception{
                JSONParser parser = new JSONParser();
                FileRead fileRead = new FileReader("match_details/"+match_id+".json");
                JSONObject jsonObject = (JSONObject) parser.parse(fileReader);
                JSONObject match = (JSONObject) jsonObject.get("match");
                long duration = (long) match.get("duration");
```

```
                long game_mode = (long) match.get("game_mode");
                String result = "";
                        if((long)match.get("match_outcome") == 2)result = "radiant";
                        else result = "dire";
                String details = match_id+","+game_mode+","+duration+",'"+result+"'";
                         for(int i=0;i<10;i++)
                         details+=","+replay.getHeroId(i);
                         playerDatabase.insertMatchData(details);
                         fileReader.close();
                }
}
```

****

```
/* Data analysis code */
/* Selecting appropriate machine learning algorithm */
import numpy
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
import pickle

def lr(features,i):
        save = pickle.load(open("../../cache/"+str(features)+"f/"+str(i)+".pkl","rb"))
        x = save['x']
```

```
        y = save['y']
        param_grid = {"C": [0.001,0.005,0.01,0.05,0.1,0.5,1,1.5,2]}
        model = LogisticRegression(n_jobs=-1)
        grid=GridSearchCV(cv=5,estimator=model,n_jobs=-1,verbose=1)
        #print grid.estimator.get_params().keys()
        grid.fit(x,y)
        print i,
        print grid.best_score_*100,
        print grid.best_estimator_.C


def svm(features,i):
        save = pickle.load(open("../../cache/"+str(features)+"f/"+str(i)+".pkl","rb"))
        x = save['x']
        y = save['y']
        param_grid = {"C": [0.01,0.1,0.5,1.5,2],'kernel':['linear'],
                      'gamma': [0.01,0.005,0.001,0.0005, 0.0001],
                      'class_weight':['balanced']}
        model = SVC()
        grid=GridSearchCV(cv=5,estimat=model,param_grid=param_grid,n_jobs=-1)
        #print grid.estimator.get_params().keys()
        grid.fit(x,y)
        print i,
        print grid.best_score_*100,
```

```
        print grid.best_estimator_.C,grid.grid.best_estimator_.class_weight



def knn(features,i):
        save = pickle.load(open("../../cache/"+str(features)+"f/"+str(i)+".pkl","rb"))
        x = save['x']
        y = save['y']
        param_grid = {"n_neighbors": [150,200,300,500]}
        model = KNeighborsClassifier(n_jobs=-1)
        grid=GridSearchCV(cv=5,estimator=model,n_jobs=-1,verbose=1)
        #print grid.estimator.get_params().keys()
        grid.fit(x,y)
        print i,
        print grid.best_score_*100,
        print grid.best_estimator_.n_neighbors


        if __name__ == '__main__':
                features=90
                time=5
                lr(features,time)
                knn(features,time)
                svm(features,time)
```

/* code to tune logistic regression */

```
import numpy
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
import pickle


features = 90
results = []
if __name__ == '__main__':
        for i in range(1,31):
            save = pickle.load(open("../../+str(features)+"f/"+str(i)+".pkl","rb"))
            x = save['x']
            y = save['y']
            param_grid = {"C": [0.005,0.01,0.05,0.1,0.5,1,1.5,2]}
            grid = GridSearchCV(cv=10 ,estimator=LogisticRegression(n_jobs=-1),
param_grid=param_grid,n_jobs=-1,verbose=1)
            #print grid.estimator.get_params().keys()
            grid.fit(x,y)
            print i,
            print grid.best_score_*100
            results.append([i,grid.best_score_*100])
            model = LogisticRegression(C=grid.best_estimator_.C)
            model.fit(x,y)
            pickle.dump(model,open("f/algorithm_lr"f_"+str(i)+"m.pkl","wb"))
           pickle.dump(results,open("/l/algorithm_lr_"+str(features)+"f.pkl","wb"))
```

```
/* logistic regression */
import numpy
import pickle

def predict(features,time,x):
        x = x.reshape(1,-1)
 model=pickle.load(open("f/algorithm_lr_"+str(features)+"f_"+str(time)+"m.pkl","rb"))
global_prediction_accuracy= pickle.load(open("lr+"f.pkl","rb"))[time-1][1]
        prediction = model.predict(x)[0]
        prediction_probablity = model.predict_proba(x)[0]
        return prediction,prediction_probablity,global_prediction_accuracy

**********************************************************************
/* predict output of live games */
import urllib2
import json
import numpy as np
from sklearn.preprocessing import StandardScaler
from algorithm_logistic_regression import predict

def num_tower_kills(towers):
        tstring = str(towers)[2:]
        count = 0
        for i in range(0,11):
```

```
                if tstring[i]=='0':
                    count+=1
            return count


def parse_match(game):
            scoreboard = game['scoreboard']
            duration = scoreboard['duration']/60
            radiant_players = scoreboard['radiant']['players']
            dire_players = scoreboard['dire']['players']
            radiant = []
            for radiant_player in radiant_players:
                radiant.append(radiant_player['assists'])
                radiant.append(radiant_player['death'])
                radiant.append(radiant_player['denies'])
                radiant.append(radiant_player['kills'])
                radiant.append(radiant_player['last_hits'])
                radiant.append(radiant_player['net_worth'])
                radiant.append((radiant_player['gold_per_min'])*duration)
                radiant.append((radiant_player['xp_per_min'])*duration)
                radiant.append(0.0)#TOWER_KILLS
                dire = []
            for dire_player in dire_players:
                dire.append(dire_player['assists'])
                dire.append(dire_player['death'])
```

```
            dire.append(dire_player['denies'])
            dire.append(dire_player['kills'])
            dire.append(dire_player['last_hits'])
            dire.append(dire_player['net_worth'])
            dire.append((dire_player['gold_per_min'])*duration)
            dire.append((dire_player['xp_per_min'])*duration)
            dire.append(0.0)#TOWER_KILLS


        radiant_18f = np.array(radiant).reshape(5,9).sum(axis=0)
        dire_18f = np.array(dire).reshape(5,9).sum(axis=0)
        radiant_tower_state = bin(scoreboard['radiant']['tower_state'])
        dire_tower_state = bin(scoreboard['dire']['tower_state'])
        radiant_18f[-1] = num_tower_kills(radiant_tower_state)
        dire_18f[-1] = num_tower_kills(dire_tower_state)
        match_18f = np.concatenate([radiant_18f,dire_18f]).reshape(18,1)
        match_90f = np.concatenate([radiant,dire]).reshape(90,1)
        scalar = StandardScaler().fit(match_18f)
        match_18f = scalar.transform(match_18f)
        scalar = StandardScaler().fit(match_90f)
        match_90f = scalar.transform(match_90f)
        return match_18f,match_90f


features = 18
```

```
res= urllib2.urlopen("https://api.steampowered.com/IDOTA2Match_570
                        /GetLiveLeagueGames/v0001/?key=ACABEB1FD8894A44B2A
                         5AB4B79209C75")
string = res.read().decode('utf-8')
json_obj = json.loads(string)
matches = json_obj['result']['games']
for i in range(0,100):
            try:
                game = matches[i]
                time = int((game['scoreboard']['duration'])/60)
                temp = time
                if time>30:
                    time=30
                m18f,m90f = parse_match(game)
                if features == 18:

prediction,prediction_probablity,global_prediction_accuracy=predict(18,time,m18f)
                else:
prediction,prediction_probablity,global_prediction_accuracy= predict(90,time,m90f)
                #if prediction_probablity[prediction]<0.7:
                #    continue
                print "Predicting matchid:",game['match_id'],
                print "  Duration:",temp,"min"
                if prediction==0:
                    print "Radiant",
```

```
                else:
                        print "Dire",
 print "has a ","%.2f%% probablity of winning" % (prediction_probablity[prediction]*100)
 print "Model accuracy at ",time," min is:","%.2f%%" % global_prediction_accuracy
 except:
       pass
**********************************************************************
/* Ranking the features */
from sklearn import datasets
from sklearn import metrics
from sklearn.ensemble import ExtraTreesClassifier
import numpy as np
from load_data_90f import load

x,y = load(30)
model = ExtraTreesClassifier()
n=5000
model.fit(x[:n],y[:n])
np.set_printoptions(precision=3)
print model.score(x[n:],y[n:])
print model.feature_importances_.reshape(10,9).sum(axis=0)

/* code for processing 18 features */
import sqlite3
import numpy as np
```

```
def getMatchOutcome(match_id,connection):
query = connection.execute("SELECT MATCH_OUTCOME FROM MATCH_DATA WHERE MATCH_ID = "+str(match_id)).fetchall()
        if query[0][0] == "radiant":
            return 0
        else:
            return 1

def load(time_elapsed):
        connection = sqlite3.connect("../../../../resources/dota2.db")
query = connection.execute("SELECT * FROM PLAYER_DATA WHERE TIME_ELAPSED="+str(time_elapsed)).fetchall()
        features = []
        lables  = []
        for row in query:
            match_id = row[0]
            match = np.array(row)
            match = np.delete(match,[0,1]).reshape(10,17)
            del_index = [1,4,7,9,10,11,12,13]
#ASSISTS,DEATHS,DENY_COUNT,KILLS,LAST_HIT_COUNT,NET_WORTH,TOTAL_GOLD_EARNED,TOTAL_XP_EARNED,TOWER_KILLS
            match = np.delete(match,del_index,1)
            radiant = match[:5].sum(axis=0)
            dire = match[5:].sum(axis=0)
```

```
                match = np.concatenate([radiant,dire])
                features.append(match)
                lables.append(getMatchOutcome(match_id,connection))
                connection.close()
                return np.array(features),np.array(lables)
**********************************************************************
/* code for processing 34 features */
import sqlite3
import numpy as np


def getMatchOutcome(match_id,connection):
query = connection.execute("SELECT MATCH_OUTCOME FROM MATCH_DATA WHERE MATCH_ID = "+str(match_id)).fetchall()
        if query[0][0] == "radiant":
                return 0
        else:
                return 1


def load(time_elapsed):
        connection = sqlite3.connect("../../../../resources/dota2.db")
query = connection.execute("SELECT * FROM PLAYER_DATA WHERE TIME_ELAPSED="+str(time_elapsed)).fetchall()
        features = []
        lables  = []
       for row in query:
```

```
                x = np.array(row)
                match_id = x[0]
                x = x[2:]
                new_f = []
                for i in range(0,17):
                        temp = 0
                        for j in range(0,5):
                                temp+=x[(j*17)+i]
                                new_f.append(temp)
                                temp = 0
                                for j in range(5,10):
                                        temp+=x[(j*17)+i]
                                new_f.append(temp)
                        features.append(new_f)
                        lables.append(getMatchOutcome(match_id,connection))
        connection.close()
        return np.array(features),np.array(lables)


x,y = load(1)
print x.shape
print y.shape
/* code for processing 90 features */
import sqlite3
import numpy as np
```

```
def getMatchOutcome(match_id,connection):
query = connection.execute("SELECT MATCH_OUTCOME FROM MATCH_DATA WHERE MATCH_ID = "+str(match_id)).fetchall()
        if query[0][0] == "radiant":
                return 0
        else:
                return 1

def reorder(radiant,dire):
        radiant = np.array(sorted(radiant,key=lambda x:x[5]))
        dire = np.array(sorted(dire,key=lambda x:x[5]))
        return radiant.flatten(),dire.flatten()

def load(time_elapsed):
        connection = sqlite3.connect("../../../../resources/dota2.db")
query = connection.execute("SELECT * FROM PLAYER_DATA WHERE TIME_ELAPSED="+str(time_elapsed)).fetchall()
        features = []
        lables  = []
        for row in query:
                match_id = row[0]
                match = np.array(row)
                match = np.delete(match,[0,1]).reshape(10,17)
                del_index = [1,4,7,9,10,11,12,13]
```

```
#ASSISTS,DEATHS,DENY_COUNT,KILLS,LAST_HIT_COUNT,NET_WORTH,TOTAL_GOLD_EARNED,TOTAL_XP_EARNED,TOWER_KILLS
                match = np.delete(match,del_index,1)
                radiant = match[:5]#.sum(axis=0)
                dire = match[5:]#.sum(axis=0)
                radiant,dire = reorder(radiant,dire)
                match = np.concatenate([radiant,dire])
                features.append(match)
                lables.append(getMatchOutcome(match_id,connection))
                connection.close()
                return np.array(features),np.array(lables)
```

TESTING AND RESULTS

7.1 Testing

The histogram when plotted for the game durations is as shown in the below figure.

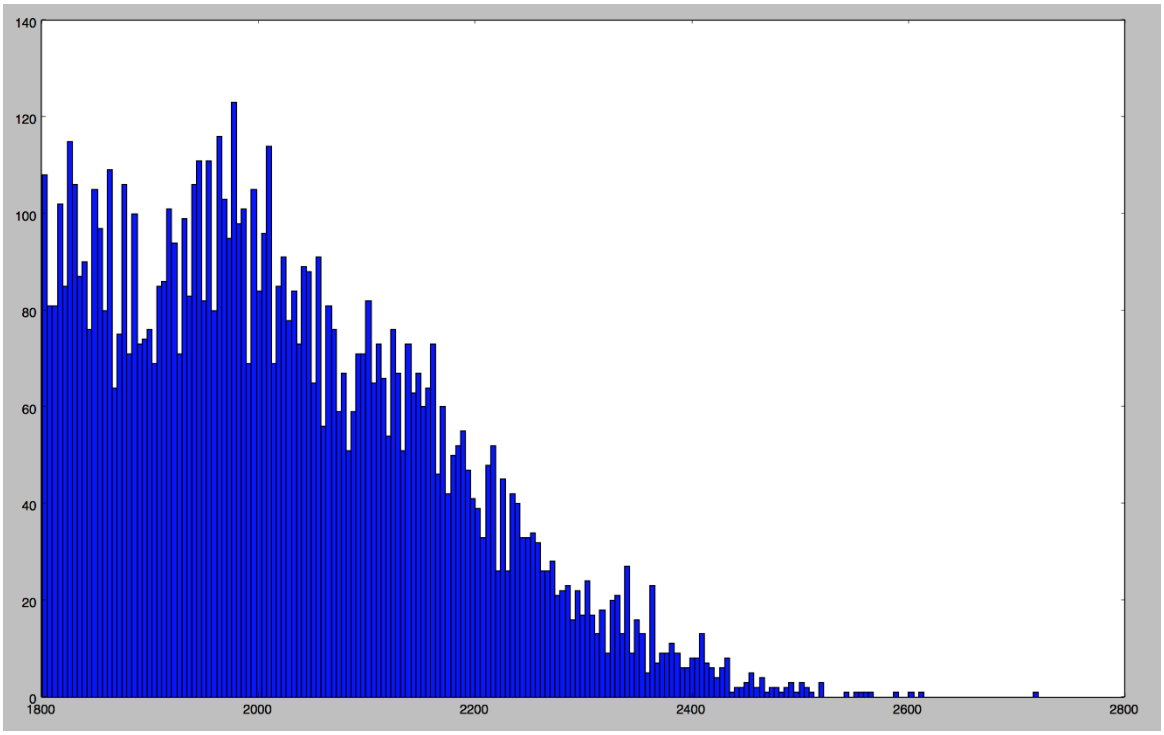


Fig 7.1 Histogram of game durations

A histogram plot of the game duration in our data set is shown below. It consists of around 8052 samples. Most of the games are around the 35-minute mark and very few games last longer than 40 minutes. this makes our model at 30 min the most optimal.

To improve the training speed of the model, all this initial processing was done and the resulting training data was cached. Once this is done we applied PCA to visualize the data set. A PCA model with Number of components = 2 was applied to reduce the dimension of the data set to 2. PCA or "principal component analysis" is used to convert a set of data points of correlated variables into a set of values of uncorrelated variables called the "principal components".

Feature Visualization at different game times with PCA (No of Components= 2) is shown in the below figure.

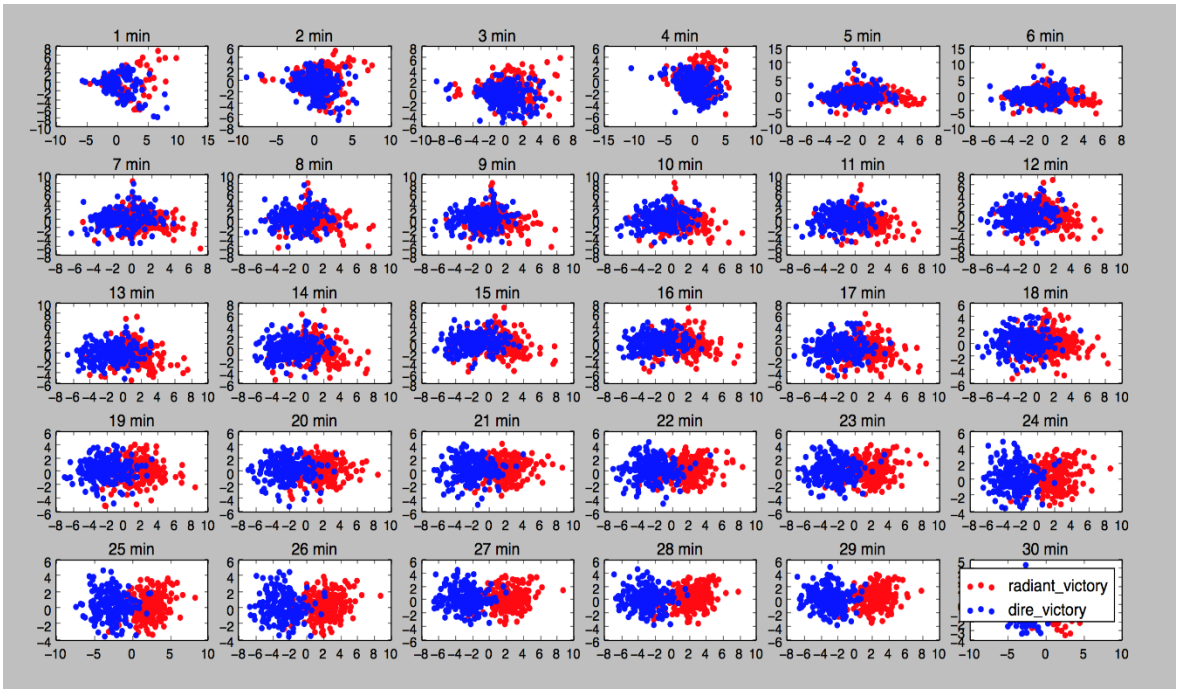


Fig 7.2 PCA analysis

The scatter plot shows all the games in our dataset, the red points represent radiant victory and blue dots represent radiant victory. As we can observe the clusters become more disjoint as time progresses. This leads to better accuracy as the game progresses.

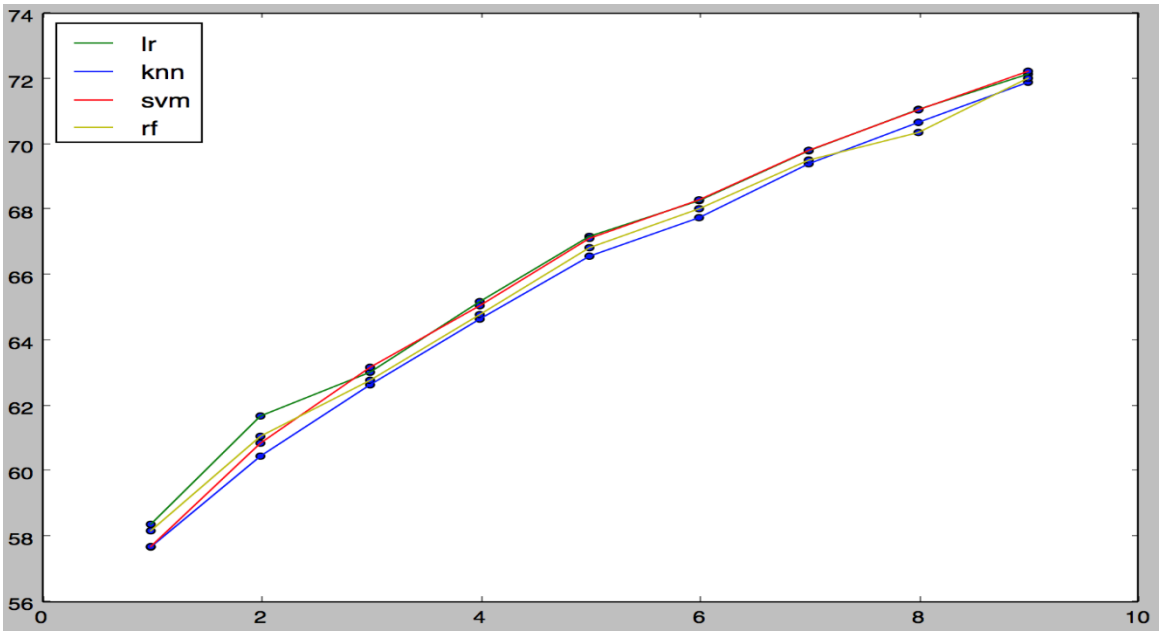


Fig 7.3 Accuracy comparison of different classification algorithms

As we can see from the above figure, the green line representing Logistic Regression performed better on an average compared to all the other classification algorithm. The only other algorithm which came close to logistic regression was Support Vector Machine but since it took Logistic regression a lot less time to train compared to Support Vector Machine, we ultimately decided to use Logistic regression. Also Logistic Regression is a probablistic model and gives us the win probablity of both radiant and dire which is useful information that we display.

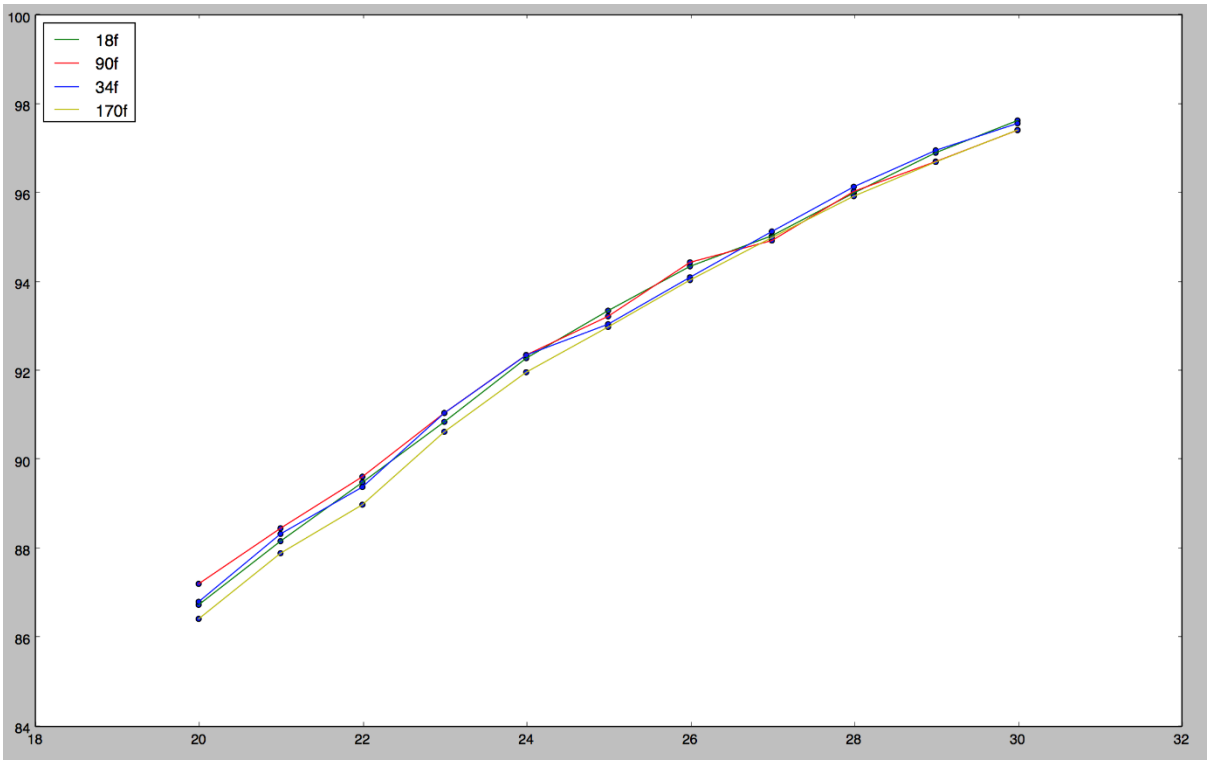


Fig 7.4 Accuracy comparison of LR for different number of features

We have also checked the accuracy of our model using Logistic Regression for different number of training features namely 18, 90, 34 and 170 features. We can see that the blue line representing 34 features gave a higher accuracy on an average. But our model which is used to predict the outcome of live Dota 2 league game uses 18 features since we can see from the above figure that the green line representing 18 features is pretty close to 34 features and the difference is insignificant. This way we are also able to reduce the "curse of dimensionality" as we now need lesser amount of data to train our model and it needs lesser time to train which is more feasible.

7.2 Results

```
Last login: Thu May 18 17:33:24 on ttys000
DWMOHAN-M-C2HZ:predictor dwmohan$ python league_predictor.py
Predicting matchid: 3191456060   Duration: 39 min
Dire has a  73.92% probablity of winning
Model accuracy at  30  min is: 97.64%

Predicting matchid: 3191496457   Duration: 14 min
Radiant has a  50.34% probablity of winning
Model accuracy at  14  min is: 78.80%

Predicting matchid: 3191508285   Duration: 11 min
Radiant has a  56.09% probablity of winning
Model accuracy at  11  min is: 74.36%

Predicting matchid: 3191511120   Duration: 8 min
Dire has a  57.15% probablity of winning
Model accuracy at  8  min is: 70.80%

Predicting matchid: 3191497459   Duration: 15 min
Radiant has a  64.26% probablity of winning
Model accuracy at  15  min is: 80.25%
```

Fig 7.5 Output of model on terminal

The above figure displays the output of our trained model on a terminal. These are predictions of live Dota 2 matches. It displays the match-id, duration, which team is going to win along with their win probability and finally the accuracy of the model at the time of prediction.

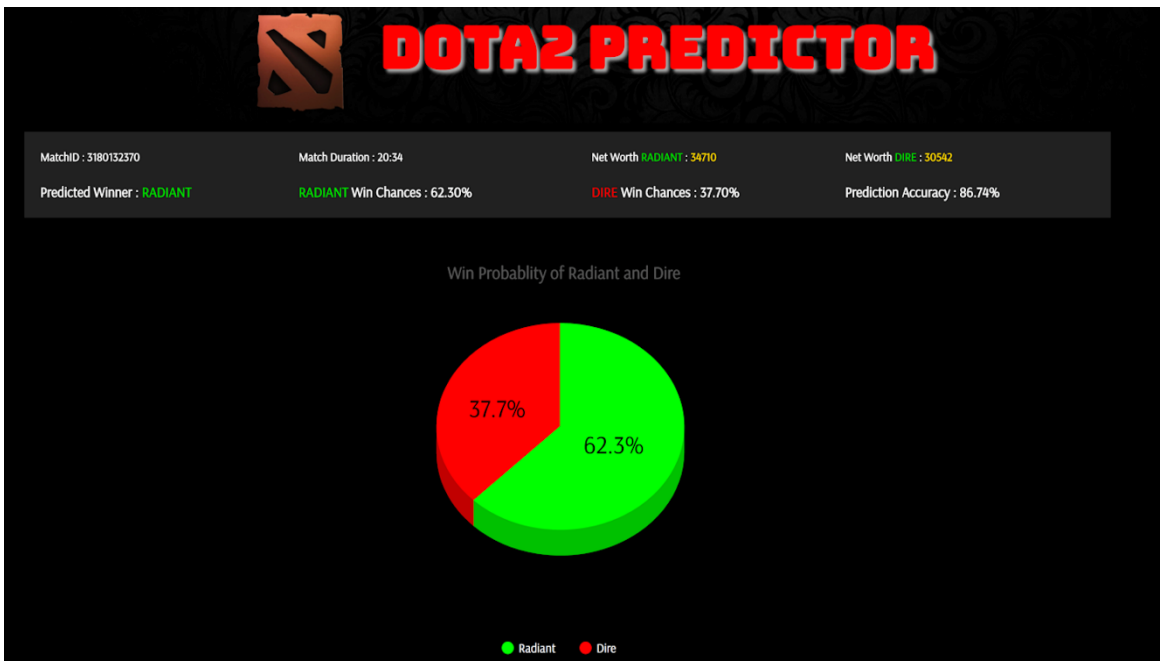


Fig 7.6 Output of model on website

The above figure shows a snapshot of the prediction app used in our website. It displays the win probability of dire and radiant (red vs green) via a pie chart. It gets this data from our trained model. Apart from the prediction, it also displays the match-id, match duration, net worth of both radiant and dire.

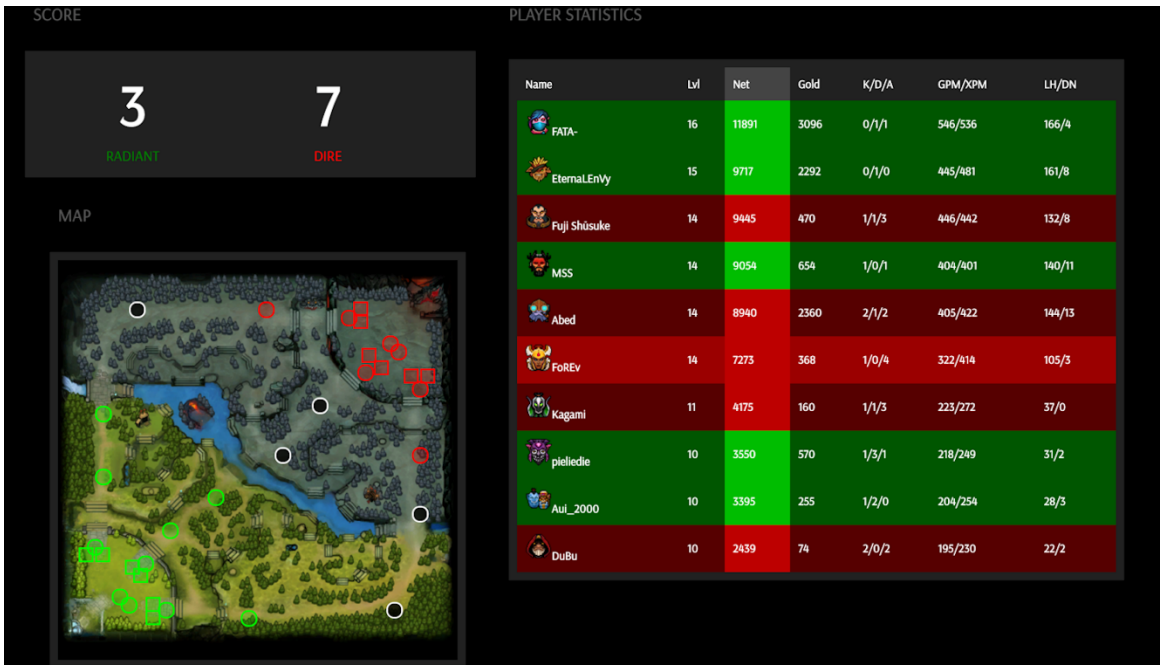


Fig 7.7 Minimap and player statistics table

The above figure shows the minimap and the player statistics table along with the net score of dire and radiant. The minimap keeps showing the status of towers. Any tower which is destroyed is instantly reflected in the minimap. The player statistics table shows various details of all 10 players like name of player, level, net worth, gold acquired, kills/deaths/assists and last hits/denies. The details in the table can be displayed in ascending or descending order of values of any of the columns.

CONCLUSIONS AND FUTURE WORK

Various machine learning classification algorithms with varying number of features were used to predict the outcome of live Dota 2 league games. From the results its concluded that partial game state data can be used to accurately predict the outcome of an ongoing game in Dota 2 in real time with the applications of machine learning techniques. From the graphs we can see that the highest accuracy was achieved by Logistic Regression at 30 min, which is 97.4 %.

A possible improvement on the dataset would be to only use replays from professional Dota 2 games, as the quality of play in those games would be higher. A higher accuracy could be achieved by using more data from replays, for example: "Stun Duration on Enemies", "Healing Done/Received". Even though hero lineups were intentionally not used in this project it would probably increase accuracy

BIBLIOGRAPHY

[1] K. Song, T. Zhang, and C. Ma, "Predicting the winning side of DotA2," Applications of Machine Learning in Dota 2, Stanford University, 2015.

[2] K. Kalyanaraman, "To win or not to win? A prediction model to determine the outcome of a DotA2 match," IEEE Instrumentation and Measurement Magazine, University of California San Diego, Vol. 7, pp 88-102, 2014.

[3] F. Johansson, J. Wikstrom, and F.Johansson, "Result Prediction by Mining Replays in Dota 2," Master's thesis, Blekinge Institute of Technology, 2015.

[4] N. Kinkade, L. Jolla, and K. Lim, "DOTA 2 Win Prediction," International Conference on Analysis of Images, University of California San Diego, 2015.

[5] A. Agarwala and M. Pearce, "Learning Dota 2 Team Compositions,", Stanford University, 2014.

[6] Dota 2 statistics," 2016. [Online]. Available: https://www.dotabuff.com/. Accessed: Oct 27, 2016

[7] K. In "Dota 2: Win probability prediction," 2016. [Online]. Available: https://inclas kaggle.com /c/ dota-2-inclass.kaggle.com/c/ dota-2- win –probability - prediction. Accessed: Oct. 27, 2016

[8] K Conley and D. Perry, "How does he saw me? A recommendation engine for picking heroes in dota 2 , CS229, Previous Projects, 2013

[9] R. Tibshirani, G. Walther, and T. Hastie, "Estimating the number of clusters in a data set via the gap statistic", J. R. Statist. Soc. B, vol. 63, pp. 411–423, 2001.

[10] L. Kaufman and P. Roussseeuw, Finding Groups in Data: An Introduction to Cluster Analysis, Wiley 1990.

[11] R. Caruana and A. Niculescu-Mizil, " An empirical c omparison of supervised learning algorithms," in Proceedings of the 23rd international conference on Machine learning, pp. 161–168, 2006.

[12] S. Abu-Nimeh, D. Nappa, X. Wang, and S. Nair, "A comparison of machine learning techniques for phishing detection," in Proceedings of the anti-phishing working groups 2nd annual eCrime researchers summit, pp 60-79, 2007.

[13] M. Mohri, A. Rostamizadeh, and A. Talwalkar, Foundations of Machine Learning. MIT press, 2012

[14] L. Breiman, "Random forests," Machine learning, vol. 45, no. 1, pp. 5–32, 2001.

[15] B. G. Weber and M. Mateas, "A data mining approach to strategy prediction," in IEEE Conference on Computational Intelligence and Games, pp. 140–147, 2009.

[16] Defense of the Ancients . [Online]. Available: http://en.wikipedia.org/wiki/Defense of the Ancients Accessed: Oct 27, 2016

[17] Steam Web API – [Online]. Available: https://developer.valvesoftware.com/wiki/ Steam_Web_APIAccessed: Oct 27, 2016

[18] Dota Picker [Online]. Available: – http://dota-picker.comAccessed: Oct 27, 2016