FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

Adaptivity in Competitive Games

Ricardo Manuel Ferreira Teixeira

Mestrado Integrado em Engenharia Informática e Computação

Supervisor: João Jacob

Supervisor: Zafeiris Kokkinogenis

November 21, 2021

Adaptivity in Competitive Games

Ricardo Manuel Ferreira Teixeira

Mestrado Integrado em Engenharia Informática e Computação

November 21, 2021

Abstract

Competitive games test the skills, strategies, and luck of multiple players vying for victory and are currently one of the most prominent types of games in the games industry. This industry has been slowly starting to adopt more data- and statistics-driven development tactics to develop their games, in order to, for example, make decisions regarding the balance of the mechanics in the games.

Balancing games is no trivial task. Game designers have to account for all the different skill levels and playstyles of players and have to come up with alternatives that cater to them, to both not lose the current player base and, if possible, to further build upon it. This is usually a very time consuming and complex process that is hard to test and balance correctly, and because of it, sometimes mistakes arise and patches become necessary. If a new beginner-friendly move is considered too powerful, most players will want to use it, and thus the game's balance will degrade, and the players might enjoy the game less. This, in some cases, even causes some backlash from the community of players, damaging the company image, and can lead to players abandoning the game, leading to financial loss to the company. In the e-sports scene, it is not unheard of characters, moves, or cards, and the likes that are banned from being used due to being too unbalanced.

An improvement to a game's balance can increase player engagement by improving player experience, which would be great for the industry.

We want to look into what factors can cause players to win in a disproportionate fashion, a staple of a badly balanced mechanic. In order to accomplish this goal we created a machine learning model using a data set from the popular game "Counter Strike: Global Offensive" (CS:GO) that predicted the outcome of a round with 78% accuracy, which in turn allowed us to look into its inner workings and understand what factors are considered more important to the classification of who wins a round, facilitating the detection of possible problems with the balance of a game, and could be a powerful tool in the belts of game developers.

A Machine Learning model that can predict the outcomes of a game can also be useful in the e-sports scene, to improve the odds in bets or a tool for event organizers to make matches more interesting by, for example, slightly help a team that is predicted to lose, making for more teeth-grinning matches. A recommendation or decision support system based on telemetry and analytics for players and coaches of e-sports teams could also be a possibility worth investigating.

Keywords: machine learning, competitive games, game balance, computer games, game adaptivity

Resumo

Jogos competitivos testam as habilidades, estratégias e sorte de múltiplos jogadores que lutam pela vitória, e são actualmente um dos maiores tipos de jogos na indústria dos jogos. Esta indústria tem lentamente começado a adotar métodos de desenvolvimento de jogos mais baseados em dados e estatísticas para, por exemplo, tomar decisões quanto ao balanceamento das diversas mecânicas dos seus jogos.

No entanto, "balancear" um jogo não é uma tarefa trivial. Os criadores de jogos têm de ter em conta todos os diferentes níveis de habilidade e estilos de jogo dos jogadores e encontrar alternativas que lhes permitam jogar eficazmente, tanto para não perder a base atual de jogadores, como, se possível, para a desenvolver ainda mais. Este é normalmente um processo muito demorado e complexo, que é difícil de testar e balancear correctamente, e por causa dele, por vezes surgem erros e correcções tornam-se necessárias. Se uma certa ação indicada para principiantes for considerada demasiado poderosa, a maioria dos jogadores irá querer usá-la, e assim o equilíbrio do jogo irá degradar-se, e os jogadores poderão desfrutar menos do jogo. Isto, em alguns casos, causa mesmo algumas reacções negativas por parte da comunidade, prejudicando a imagem da empresa, e pode levar a que os jogadores abandonem o jogo, levando a perdas financeiras para a empresa.

Uma melhoria do balanceamento de um jogo pode aumentar o engajamento por parte dos jogadores ao melhorar a experiência dos jogadores, o que seria ótimo para a indústria.

Nós queremos olhar para que fatores podem fazer com que jogadores ganhem com muita facilidade, o que é um indicador forte de alguma mecânica mal balanceada. Com este objetivo em mente, criamos um modelo usando algoritmo de aprendizagem de máquina usando um conjunto de dados do jogo "Counter Strike: Global Offensive" (CS:GO) que é capaz de prever o resultado de uma ronda com 78% de precisão, o que nos permite olhar para como funciona e perceber que fatores influenciam mais a classificação de quem ganha um ronda, facilitando a detecção de de possíveis problemas com o balanceamento de um jogo, o que pode se tornar numa poderosa ferramenta para os criadores de jogos.

Um modelo de aprendizagem por computador que consiga prever os resultados de um jogo pode ser muito útil em e-sports, para melhorar as chances das apostas ou como uma ferramenta para organizadores de eventos para tornar os encontros mais interessantes ao, por exemplo, ligeiramente ajudar uma equipa que esteja previsto que perca, criando jogos mais empolgantes até ao final. Um sistema de recomendação ou de apoio à decisão com base em telemetria e análise para jogadores e treinadores de equipas de e-sports também seria uma possibilidade que vale a pena investigar.

Keywords: aprendizagem por computador, jogos competitivos, balanceamento de jogos, jogos de computador, adaptatividade de jogos

Acknowledgements

Throughout the writing of this dissertation, I have received an incredible of support and assistance. I wish to thank the various people who contributed in one way or another for the success of this dissertation, either directly via advice or indirectly by just being present in my life. Without them, doing this dissertation would be far more challenging.

Before anyone else I would like to thank my supervisors, Prof. João Jacob and Prof. Zafeiris Kokkinogenis, for their constructive and incredibly valuable guidance, even when times were tough. Their advice and their availability throughout the planning and development of this research work proved to be of great help to the results of this project. But most of all, I am thankful for your faith in this dissertation and in me.

I would also like to thank my closest friends, Ricardo Moura, Fábio, Ana Silva, Xavi, Miguel and Ana Rangel, for giving me the strength to continue when I grew weary, for believing in me even when I didn't, for giving the much needed distractions so I could rest from this work, and last but not least for the ones that walked the extra mile and helped me through a lot of the problems and doubts that I faced.

Finally, I want to thank my family for all the continuous love and support they gave me. My aunt and my uncle for having to deal with me every day, my cousins who didn't do much but were simply there, my parents for giving me the opportunities and experiences that have made me who I am, my grandparents for always showing how proud they are of me and for encouraging me in every single way, not just during the development of this dissertation but throughout my life.

Thank you from the bottom of my heart for shaping the person I am today. This thesis was only made possible by all of you.

Thank you,
Ricardo Teixeira

“(...) Plans are useless,
but planning is indispensable.”

Dwight D. Eisenhower

“I’m a great believer in luck,
and I find the harder I work, the more I have of it.”

Thomas Jefferson

Contents

List of Figures

List of Tables

Abbreviations

AI	Artificial Intelligence
CS:GO	Counter Strike: Global Offensive
CT	Counter-Terrorist team
DT	Decision Tree
FPS	First-person Shooter
LR	Logistic regression
LSVM	Linear Support Vector Machine
ML	Machine Learning
NB	Naive Bayes
RF	Radom Forest
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
T	Terrorist team

Chapter 1

Introduction

Contents

Video games are one of the many options of entertainment available nowadays. The video games industry has seen steady and steep growth over the last years, seeing a compound growth rate of 8.21% between 2014 and 2018, with a market size reaching \$24.2 billion in 2019 and is expected to reach \$28.1 billion in 2023 [52]. However, some estimates, including more indirect sources, consider the global market value to be at \$175.8 billion in 2021 [37]. Amid this growth, competitive multiplayer games appeared as a trendy type of game. These games' popularity became so large that they and their player base even spawned the new industry of e-sports, where tournaments exist with prizes in the millions of dollars and professional players can exist.

Competitive games pit two or more players against each other in a fight for a final victory while testing their skills, strategies and luck. However, for a game to be enjoyable to play, the level of challenge faced by the player needs to be adequate: If it is too high, the player will be frustrated, too low, and the player will become bored. However, in competitive games, the challenge faced is from other players. If players discover an optimised strategy with a very high chance of winning, players will tend to gravitate towards that strategy. Those players who do not use it have a more challenging time competing, if not making it outright impossible to win. The effect of this is that the game will become much less varied, as everyone will try to do precisely the same things. This effect might make the games less entertaining for all players, as they have to stick to the same strategies if they want to win, making the players who execute them better the most successful.

In single-player games, this level of challenge faced by a player is managed by a plethora of different techniques, from self-profiling and choosing a difficulty level to the usage of artificial intelligence to modify the challenge level automatically. The balancing problems of a particular action is more easily detected via play-testing, and the consequences of a poorly balanced game tend to be less severe.

In competitive games, however, most of these known solutions cannot be applied in the same manner. Therefore, these games must be well balanced so that all players can have a fighting chance, but at the same time not have a single optimal approach that every player should take, as that would make the game less interesting. If the matches become less enjoyable, the game will lose the support of its player base, losing potential revenue. As examples of this happening, it is not unheard of that certain characters, moves, or cards and the likes become banned from being used during tournaments for being too powerful and therefore considered an "optimal strategy" and therefore unbalanced.

1.1 Context and Motivation

A game designer's job is not an easy one when it comes to balancing competitive games. To cater to the biggest audience possible and keep the current player base, game designers have to implement options for players of all skill levels to compete fairly. This process takes a long time and dedication to get right due to its exponential complexity and because it is challenging to test if the game is well balanced or if there are any problems. Due to these challenges, mistakes might sometimes arise, and corrections or patches become a necessity.

For example, game designers will often create new beginner-friendly moves, characters, abilities, or weapons that are easier to use than the usual counterparts to cater to the more beginner audience. If a particular beginner-friendly move is considered too powerful, most players will want to use it, and thus the game's balance will degrade, and the players might enjoy the game less. In some cases, this even causes some backlash from the community of players, damaging the company image, and can lead to players abandoning the game, leading to financial loss to the company. A clear example of this happened in Counter-Strike: Global Offensive, a First-Person Shooter (FPS) game published by Valve Corporation. The R8 Revolver, a newly introduced pistol, created an outcry from the players due to some problems with the balancing of it: This pistol was able to kill players with one shot and had almost 100% accuracy while moving. It is understandable why players were unhappy with the new weapon, and some even threatened to stop playing until this issue was resolved. Valve reacted quickly by reducing the weapon's damage, increasing spread and cooldown between shots, only days after the release of the weapon [1, 9].

Thus, this work's motivation becomes clear: to find novel ways of detecting these balance issues in competitive games, using data extracted from regular gameplay.

1.2 Research Questions

The following research questions were defined to guide the research work and development needed for the proposed solution, making sure it has a well-defined scope:

- How to improve game and player balancing in a competitive setting using machine learning?

- Is it possible to detect issues in balancing using a machine learning approach?

The first question is related to the design and implementation of the proposed solution, which requires research on the current state of the art. The second question concerns the implementation and validation of the proposed solution.

1.3 Objectives

Objectives were determined to provide the thesis with goals to move towards:

- Research common ML techniques for game balancing;
- Design a methodology that fosters supervised learning to help detect possible balancing issues on competitive games;
- Implement a case study based on a previously developed competitive game;

1.4 Document Structure

This document is divided into several chapters for easier reading and organisation. The current chapter, Chapter 1, introduces the problem, gives the contextualisation and provides the motivation and objectives for this work. The next chapter, 2, presents a state of the art review, where related research areas are analysed, starting with the fields of Machine Learning, Game Balancing, and how Machine Learning has been used to improve Game Balancing

Chapter 3 describes the methodology that we used in this thesis in more detail, presenting the process we have followed to achieve the results presented in chapter 4.

Finally, chapter 5 concludes what was presented in this document and future work, along with a summary of what was learned from the process.

Chapter 2

State of the Art Review

Contents

This chapter aims to overview the areas of study related to this thesis' subject while analyzing their history. The State of the Art review starts with section 2.1, where the current techniques in the field of Machine Learning are reviewed. Afterwards, in section 2.2 we review various Game and Player Balancing techniques, including some machine learning applications for game balancing in section 2.2.5. Finally, in section 2.3, a summary of the state of the art review is provided to highlight some of its main characteristics.

2.1 Machine Learning

Machine learning (ML) is a branch of artificial intelligence that systematically applies algorithms to synthesize the underlying relationships among data and information. For example, ML systems can be trained to convert acoustic information in a sequence of speech data into semantic structure expressed in the form of a string of words on automatic speech recognition systems (such as iPhone's Siri) [15]. Machine learning can be divided into four types: supervised, unsupervised, semi-supervised, and reinforcement [23].

2.1.1 Supervised Learning

In supervised learning, the dataset is a collection of labelled examples. The goal of a supervised learning algorithm is to use this dataset to produce a model that takes a certain collection as input and outputs some label for that collection.[23].

2.1.2 Unsupervised Learning

In unsupervised learning, the dataset is a collection of unlabelled examples. Once more, the goal of a unsupervised learning algorithm is to use this dataset to produce a model that takes a certain collection as input and either transforms it into some other collection, or into a value that can be used to solve a practical problem.[23].

2.1.3 Semi-Supervised Learning

In semi-supervised learning, the dataset contains both labeled and unlabeled examples. Usually, the quantity of unlabeled examples is much higher than the number of labeled examples. The goal of a semi-supervised learning algorithm is the same as the goal of the supervised learning algorithm. The hope here is that using many unlabeled examples can help the learning algorithm to find (we might say "produce" or "compute") a better model. [23].

2.1.4 Reinforcement Learning

Reinforcement learning is a subfield of machine learning where the machine "lives" in an environment and is capable of perceiving the state of that environment as a vector of features. The machine can execute actions in every state. Different actions bring different rewards and could also move the machine to another state of the environment. The goal of a reinforcement learning algorithm is to learn a policy. [23]

A policy is a function (similar to the model in supervised learning) that takes the feature vector of a state as input and outputs an optimal action to execute in that state. The action is optimal if it maximizes the expected average reward. [23] Algorithms such as Q-Learning [56] and proximal policy optimization (PPO) [45] are well known algorithms in the reinforcement learning area. Another case is AlphaZero, a general reinforcement algorithm that mastered chess, shogi and Go by playing alone, with no human input [48].

2.2 Game and Player Balancing

Since our objective is to detect possible balancing issues with games, we believe it is important that first we research what techniques are already used to attempt to correct balance issues, many of them in an automatic manner, so that we may learn something about how they operate and possibly guide our decision-making process in the rest of our work.

In designing a game, there are multiple forms of balancing that can be applied to improve the experience. Usually balancing a game for an individual player means adjusting the game to experience an adequate challenge level. Suppose a player faces a challenge level that is way above its ability. In that case, the situation becomes upsetting and frustrating. Otherwise, if the challenge is too low, then the player might eventually grow bored of the game. Keeping the challenge level adequate to the player expertise will keep the players in the "flow zone" [49], and games that can keep players in this state are more enjoyable and gripping.

When it comes to competitive multiplayer games, however, things get much trickier. Players can have very different skill levels, and since most of the challenge a player faces in competitive games is, by its very nature, defined by the players one is up against, the challenge level is hard to control. With traditional game balancing methods, the expert players will always win over the novices, usually by a landslide, leading to a worse overall experience for both sides than what could have been achieved by a more intense fight. In some games, player balancing techniques, such as matchmaking, are used to balance the players themselves instead of the game, so that players' skill levels are similar, and the games become more engaging than before. These techniques will be covered here as well for they might bring some insight into this work.

2.2.1 Difficulty Adjustment

Difficulty adjustment is the process through which the games try to adjust the difficulty faced by the players by certain parameters on how the game works. Difficulty adjustment in games is most commonly performed in a static manner, hence the name static difficulty adjustment. With this type of difficulty adjustment, the player selects among certain options for what level of challenge they want to face. The most common execution of this is through the implementation of "difficulty levels", commonly selected at the start of the game, but a more granular ways of personalising the difficulty settings also started to appear. A popular example of these personalised difficulty settings is the game *Celeste* [34], a game filled with difficult platforming challenges that require a lot of skill to execute. Celeste allows the player to lower the normal difficulty of the game by enabling the "Assist Mode", where the game can be slowed down, or certain mechanics such as stamina or invincibility can be switched on and off, allowing for a wider audience to be able to play and enjoy the game. This type of difficulty adjustment does have some issues however, namely the fact that some self-profiling is required from the player, and it can also lead to cases where the level of challenge is not at all adequate to the player's skill level [31]. In competitive multiplayer games, however, self-profiling is very often not even a possibility, and therefore this path of work has very small chances of being helpful to our objectives.

2.2.1.1 Dynamic Difficulty Adjustment

Compared to the more static difficulty adjustment approach, dynamic difficulty adjusts the game's difficulty according to the player's performance as the game progresses. The items in the *Mario Kart*™ games, the most recent being *Mario Kart*™ *8* [38], follow those rules, by providing better items to those falling behind in the race, so that they have better chances of catching up. This makes the races more back-and-forth up until the end, creating a more enjoyable experience to the players involved.

Techniques like these, however, if not well implemented, can be highly noticeable to players, and some players might be able to exploit this better than others, leading to players feeling like they were cheated [31, 16].

BinG is a framework that collects and processes data provenance, to allow the development of different game models to be used externally to the game. A study of a game with dynamic difficulty adjustment based on this framework proved to reduce the discrepancy of player performances considerably [26].

2.2.2 Matchmaking

The concept behind matchmaking systems is that the skill of a player should be relatively similar in a series of games, and therefore players could be comparatively ranked against one another like in the ELO rating system, which is used in chess [27]. This ranking makes it possible to make the players of similar skill levels play together, making the matches more exciting. For competitive video games we start start seeing a lot of ranking algorithms based on newer algorithms, such as the TrueSkill algorithm, a bayesian skill rating system [29], and later a new improved version (TrueSkill 2) appeared [35]. TrueSkill was used to rank players in many popular games such as *Halo 3* or *Forza Horizon 7*, and the improved TrueSkill 2 was also used in many newer games [35].

Popular games such as *Valorant*, *Rocket League* or *Apex Legends* have ranking and matchmaking systems like these in place; however, these systems often have to make compromises based on the available players and do not account for other problems such as temporary variations in a player's performance.

2.2.3 Asymmetric Roles

In some team games, certain characters can fill specific roles and have different levels of difficulty of use. This can lead to some natural balancing when particular roles have a natural advantage over others, similar to rock paper scissors. These games need to have some thoughtful design so that every role needs to be useful in some way so that no role becomes irrelevant.

The downside to this kind of game design is that the more options there are, it becomes increasingly hard to balance correctly, and game balancing issues might appear if specific roles are not fulfilled by the players, forcing them to play roles they might not want to. Games like *Overwatch* or *Evolve* are good examples of asymmetric roles. Especially in *Evolve*'s case, the game developers admitted that their very different roles were next to impossible to balance [57].

This approach lead to new approaches being developed, such as the usage of Monte Carlo Simulations to analyse the relative impact of game actions, and iteratively adjusts attributes of the game actions till the game design is balanced by approximation [17].

2.2.4 Aim Assistance

In one of the most popular genres of games, the First-person Shooter, a less common approach is to do player balancing by making the less skilled players have an easier time landing their shots. This can appear in different ways, such as bullet magnetism (i.e. bending the paths of bullets a bit to a nearby enemy), change the projectile's size, explosive or area of effect weapons, and the likes. Not all of these techniques work as well as each other, mainly when applied in real-world scenarios, and some of them are too noticeable for players. However, if the correct techniques are applied well, weaker players will be able to shoot their desired targets more often. This has been proven to help balance a competitive multiplayer FPS game [54, 55]. One of the common fears of using these techniques in games is that players will become overly dependent on the assistance and not that this would hurt their aiming skill learning progress. This claim has been disputed however [28], suggesting that the value added by aiming assistance would exceed the possible concerns about player aim improvement.

2.2.5 Machine Learning in Game Balancing

There has also been a plethora of research on the usage of machine learning to perform balancing actions on games, from better and more diverse artificial intelligence (AI) for computer players, to predictive models that classify a players skill level.

An example of the process of creating new better AI for computer players can be seen in the work of Olensen et al., 2008 [40], where they used the Real-Time Strategy (RTS) game *Globulation 2* to develop a successful application of the genetic algorithms NEAT and rtNEAT to the AI players. This work also provides insights into what attributes were more and less relevant to the artificial neural networks' final results. It was also suggested that rtNEAT would be better suited for AIs in games with interaction between players and opponents, such as First-Person Shooters (FPS), fighting games or 2D action games [40].

Fighting games have shown significant research in this area: One such example is a work that uses Unity's ML-Agents Toolkit to run the test scenario, a fighting game between two non-player14agents that use a Reinforcement Learning, Proximal Policy Optimization (PPO) algorithm [39].The PPO implementation follows a Deep Reinforcement Learning approach To tackle the complexity of the environment, and showed promising results, at least when facing other non-human players. Other works [14, 42] use a fighting game as its test case, and they created an adaptive reinforcement learning agent, using Q-Learning, that was able to adapt to the actions of other agents, maintaining similar performance levels to the opponent, interleaving wins and losses, no matter the real skill level of its opponent. This work uses an heuristic as its challenge function that is mostly based on the game score, and it can be adapted to other types of games with the same sort of scoring mechanisms.

Dynamic difficulty adjustment was also achieved by the use of neural networks with multi-layer perceptron (MLP) architecture [46] trained through evolutionary algorithms to achieve better adaptation. The system learns and adapts online to the style of the player's in-game behavior based on the neuroevolution using additionally the base of catalogues, allowing for quick switching of neural network weights when skill levels fluctuate rapidly, and collect the statistical data about player behaviours.

A game that has been in the spotlight for its revolutionary usage of AI is the series Left 4 Dead and its famous AI Director, the artificial intelligence model that controls the games pacing and difficulty, by placing enemies in varying positions and numbers based on each player's current situation, status and skill levels. It also creates tension and moods with emotional cues, such as visual effects, dynamic music and character communication. Items such as health, ammo and weapons, as well as special, more difficult enemies, are also controlled by the AI Director [19, 6, 24]. The creation of this AI took lessons from one of the company's other game series, Counter-Strike, as for the way the AI should model the desired tension curves [19]. The success of this game is a statement to how powerful can Artificial Intelligence be in creating more dynamic experiences for players, increasing replayability and player retention. This AI director has inspired more research works, such as the usage of Defeasible Logic Programming to create an argumentative AI Director for dynamic map generation in a roguelike game called HermitArg. The architecture of the game introduces smart items with defeasible information to be analyzed in a dialectical process [13].

Facial expression recognition has also been considered as a tool for extracting information for game difficulty balancing [36], improving the player experience by adjusting the game difficulty according to what the player is feeling.

Balancing the options available to players in a way that does not sacrifice variety and viability is a tough problem to solve. A solution to this issue was proposed by Pfau *et al* for the usage of deep player behaviour modeling to represent a population of players in the massive multiplayer online role-playing game *Aion*, and in turn generate individual agent behaviours. This work demonstrated significant balance differences in opposing enemy encounters and showed how they could be regulated [41].

A paper that later would become very important for the definition of the proposed solution is a work that was able to predict a player's previous experience with first person shooter games with just gameplay input of a FPS game, with 76% accuracy, from just a minute of gameplay data. [21]. The technique used by this paper for the model was the Random Forests, a method for classification and regression, constructed from decision trees. A following work by the same authors as the one before is also able to predict a player's skill level in a FPS game, not just player experience, from just the first thirty seconds of mouse and keyboard input and with high accuracy [22]. Together both of these works showed that ML can be used for skill prediction in a first-person shooter, and gave us an indication as to what features could be important for planning the proposed solution.

There are also some recommender systems that have started to appear that make use of player information to improve their recommendations. A recommender system for Player-vs-Player (PvP) encounters was designed for the game Destiny, a popular massively multiplayer online game, using multiple parameters from character stats to weapon playstyles to characterized some behaviour aspects [47]. A commercially available AI-based assistant, called SenpAI.GG, was also created for the game League of Legends, which analyses team compositions to suggest champion picks and which items to buy over the course of the match, and also providing some insight on the progress of the match, with the goal of improving player performance [10].

2.3 Summary

In section 2.1, there was an overview of various concepts from the area of machine learning, to better understand their usage in the sections to come and possible advantages in the proposed solution. Afterwards, in section 2.2 and respective subsections, we made an overview of the multiple techniques that employed for game balancing and player balancing, with the objective of learning how balancing issues are being corrected, some in an automated manner. We saw the advantages of dynamic difficulty adjustment over static difficulty adjustments, the current problems with matchmaking systems and their weaknesses. We also saw the possibility of having multiple different and complimentary roles in a team, so that players can play as the role they are more comfortable with, and the different aim assistance techniques that exist for the example of player balancing in a First-person Shooter. In the subsection 2.2.5, we also explored various approaches of previous works that used ML to balance the game or its players, and try to get some insight from that work.

Chapter 3

Methodology

Contents

In this chapter, we will describe in more detail the resources that were used (including the chosen game and data set) and the approach that was used to process the data set in order to obtain interesting results.

3.1 Resources

In this section, we will present an overview of the resources we used for the implementation of our case study. We will take a look into how we chose the game that is the basis for the study case and the reasoning behind the choice of the used data set. We also describe the machine learning resources that we used in our implementation.

3.1.1 Game and Data Set Choice

In order to start, we have to at first obtain a data set from which we can extract useful information. Since we had no data set of our own, we had to pick a public data set from an existing game. Since the aim of this research should be applicable to the largest amount of games as we can, we wanted to use a game that followed a few conditions:

- Should not have very complex mechanics / should be relatively simple to understand. A very complex game increases the noise in the data and can have more complications (e.g. interactions between mechanics, false correlations), hindering the process.
- Should be from a popular "genre". A simple game from a popular genre makes the process and this research easier to think about and more readily applicable to a broader type of games.
- It would be even better if the game had an actual following in e-sports or streaming, as it could be interesting to think about the possible impacts of the work on spectators.

Given these criteria and after considering a lot of games (for example, Starcraft and Defence of the Ancients 2 - DOTA 2), we ended up choosing the game Counter-Strike: Global Offensive (CS:GO), which is a very popular team-based first-person shooter, where two teams (Terrorists or T, and Counter-Terrorists or CT) face against one another to accomplish specific objectives in order to win enough rounds to win a game. CS:GO has many game modes, but we are, however, most interested in the classic competitive game mode, as it is the one that has the most significant following and even has very active e-sports and streaming scenes, which bolsters its position. Besides this, the competitive mode includes a ranking system in the match-making so that players of very different skill groups would not play together and skew the data.

In the competitive mode, both teams, with five players each, face against one another with different objectives. There are two separate "bomb sites" on each playable map, and the Terrorist team has one bomb to plant in one of them. After a bomb is placed, a forty-second timer starts ticking down, which upon finishing, the bomb explodes, granting the win for that round to the terrorist team. The CT team can either stop this by preventing the T members from planting the bomb or defusing it before the timer runs out. If a team kills all enemy players from the other team, that team wins the round as well (with the exception that if the dead T players have planted the bomb, the CT players still have to defuse it before the timer runs out in order to win).

There is also an economic system in place, where players have to spend money to buy better equipment and weapons. Money is awarded by killing players from the opposite team, planting and defusing the bomb, and at the end of the round, the winning team usually gets more money than the losing team. The matches go on until a team scores 16 points, or if both teams score a total of 15 points, the match ends either in a tie or continues to overtime until there is a two-point lead by one of the teams, and in this case, teams have a considerable increase in money rewards to buy equipment.

We consider this game to be relatively "simple" because most of the game's mechanics involving physical skill - such as player movement and gun-play, are very close to the current standards of the first-person shooter genre. The complexity of this game comes mostly from cognitive skills - such as tactical thinking for positioning and strategy, and from the economic system, where players have to manage their funds across different rounds. There is also a relatively large amount of different weapons of different value that can be bought, all with their own properties to suit different playstyles.

After the game was chosen, a search for a data set to use had to be conducted. Fortunately, there was a large amount of very different data sets available online, especially coming from websites such as Kaggle[1]. However, we ended up using a data set from a competition that occurred during the Summer of 2020, called "CS:GO AI Challenge" [2, 3] whose challenge was to predict in-game situations and state likely outcomes [2].

3.1.1.1 Data Set Description

This data set contains data from around 700 demos of high-level competitive tournament play in 2019 and 2020. The matches are all using maps where the objective revolves around the planting or defusing a bomb, depending on what team the players are on. Certain cautions were taken to filter the warm-up rounds and restarts from the data, and for the remaining live rounds a round snapshot of the game was recorded every 20 seconds until the round was decided. [5]

The data set contains the following information:

- **Map and Game Version** - the map where the round was being played, as well as the patch on which the game was played;
- **Current Score** - Current round scores for both the Terrorist (T) and Counter-Terrorist (CT) teams in this game;
- **Round Status** - Describes in what stage of the round this snapshot is from, and also how long this round can last. The options for what stage the round is in are:
  - Freeze Time: the round did not start yet; players are frozen and can use this time to purchase equipment and discuss strategies;
  - Normal: the round is now running with players alive on both sides. The bomb at this point was not planted;
  - Bomb Planted: the round is still undecided. The bomb was already planted, but still not defused or exploded;
  - Slack Time: At this point, the winning team of the round has been decided, either by the explosion or the defusal of the bomb, or by a team having all of its players dead before a bomb plant occurred.
- **Alive Players** - A list of all the alive players from both teams, with information about them such as health, armour type and amount, money, their equipment and weapons, as well as a list of some known positions throughout the round up to that point.
- **Active Smokes and Molotovs** - A list of all the active smoke grenades and molotovs/incendiary grenades on the map, along with their positions

[1] www.kaggle.com/

- **Player Kills** - A list of all the player kills from both teams, with information about what weapon was used, the position of the attacker and defender, and the teams of both players (since friendly fire and suicides are possible).
- **Round Winner** - It states what team ended up winning the round of this snapshot, clearly intended that this should be a target variable.

As seen by the description, this data set does not contain information on who each player is, or what specific player executed what actions. The information we have this way is more "general" as the team's composition at the moment, more than the individual performances of the players. This detail comes with both advantages and drawbacks, some of them becoming apparent when considering the objectives of this work.

The lack of individuality in the player data can "blur" the line between a game of five versus five and a single team versus another. This effect can be used to generalise the approach we take so that it can more easily be applied to other types of competitive games, for example (1 vs 1). As this work is intended to showcase a process of analysing competitive games for predicting results and detecting possible imbalances of a "generic" competitive game, this data set can therefore be used as a decent "ground-level" that can be expanded upon in the future for more particular examples.

However, by not including information about particular players, or even not having any guaranteed connection between screenshots, we are not able to have any information about the players' previous performances, so we are losing significant potential information that could allow for a better prediction of outcomes. One such example of data that could be useful is a classification of the players' "skill levels" - a factor considered to be crucial for who wins the game. These factors, however, tend to be considered more in the domain of match-making algorithms, so the differences in skill could be minimised at the start of a game.

3.1.2 Machine Learning Resources

We mainly used the Scikit-Learn[2] library in order to train and test our models, using their implementations of different classifiers and other valuable features for feature engineering purposes. Jupyter Notebook[3], although not directly a ML resource by itself, was also used to provide a more interactive and streamlined approach to the development of this work.

3.2 Approach

After we have chosen an appropriate data set, work begins on the approach that we would take to make our models. At first, we had to be able to load the data into a useful format and start with the cleaning process - detecting faults in the data, impossible situations, and other similar issues so that faulty data would not interfere with the model. Afterwards, the feature engineering process began, where features would be extracted from the data, go through proper encoding and through

[2]www.scikit-learn.org
[3]www.jupyter.org/

some possible selection methods to choose only the most valuable features for the model. With the processed data and all extra features in hand, we were able to extract some statistics on this data set that we have, revealing some interesting information. Afterwards, we can start to plug in all this data into the machine learning algorithms to start having some results for our work. It should be noted that these processes do not necessarily occur in linear order, and revisits to previous work might be needed according to the needs of the project.

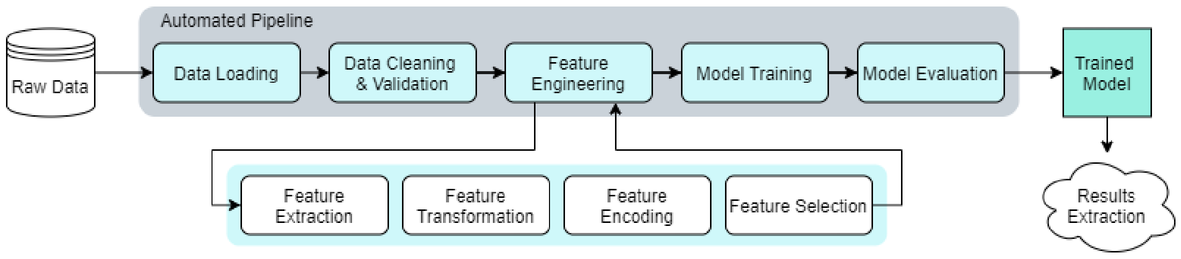


Figure 3.1: Automated Pipeline

To execute all of these steps in quick succession, we created an automated pipeline, a short overview of which can be seen in figure 3.1. This pipeline performs all of the steps, sending the data output from each phase right in as an input of the next phase. At the end of this pipeline, the models are created, hopefully allowing for some insights to be gained into the problem.

3.2.1 Data Loading and Cleaning

With the data set in hand, the first order of business was to load the data set into a usable format. The raw data in the data set was in JSON format, and we would need to convert it into a more usable format, such as a pandas' DataFrame. Nowadays, with the amount of data that is possible to collect from our world, it is increasingly common to have a large volume of data partitioned through various separate files. This is what happened with this data set, where its data was spread out over 25 different files, most over 30 MB in size. For simplicity, we chose to concatenate the data from all files into a single usable DataFrame.

With the data all loaded into memory, it was time to start the process of tidying this data. We can go through this in many ways, but we will share the things that we had to take mostly into account in the following subsections.

3.2.1.1 Missing Values

After the data was loaded, the process of cleaning the data was started. The first thing that is usually done is to check for missing data and what to do when some of it is found. In this case, our data set did not have missing values anywhere, so this step was not performed, but this step should be mentioned for completeness' sake. There are two main ways of dealing with missing values like this: either by removing the samples with missing values or filling them up. Removing the rows with missing values is one of the simplest ways of dealing with missing values; however, it may not be an ideal strategy if the data contains a large portion of missing values. If we do end up dealing with missing data this way, we have to be careful that the data we are deleting does not include information that is used in other rows of the data. On the other side, one can choose to fill out these missing values. This step can be done in very different ways depending on the data that we possess. For example, if the data is of a categorical type, we can create an entirely new category for the missing values or fill it in based on different categorical data. However, if the data is of numeric nature, the mean and median can be used (along with other statistical approaches) to rectify this issue [20, 50].

3.2.1.2 Detect Data Errors

As we started to look into the data that we had, we noticed that some rows had data that was impossible to be correct under normal circumstances. Examples of this issue include rows where one team had six players, where the game only allows five players on each team, or where teams started the match with incredible amounts of money, not the standard $800 per player. Such rows are not in line with the rules of this mode, and therefore we had to cut them so they would not influence the models in any way. The way that we detected that some of these errors were inside the samples was mostly by looking for outliers (a data point that differs significantly from the other observations) in the data. Outliers are sometimes good indicators of errors or something that can have gone wrong, but not always, as an outlier might be a valid data point. Since it is data taken directly from a game, assuming the data collection system is good enough, the data we receive should have minimal errors and might indicate some weird strategy that a specific player has, or the effect of a glitch/bug in the game. For example, in the case of a team having six players being reported where only five is specifically allowed, the only case I could find where this was possible was related to players leaving and reconnecting to a match [4], however since this probably is not the intended behaviour, we did not consider these rows from our data set.

3.2.1.3 Converting Data Types

A given data set can contain various different data types. The structure of the data that we have is essential, as an inadequate structure can, at best, lower performance and, at worst, turn our model impossible to train. We may find various data types such as:

- Categorical data, for data that has a limited number of distinct options. This type of data has two sub-types:
  - Ordinal data, for data that can have a clear order of importance. The position of a player in a racing game could be interpreted as such.
  - Nominal data, for data that does not have any relationship with each other. The map that the CS:GO round was being played on is an example of this type of data. Possible

[4] 6 players in competitive? - Reddit post

values include 'de_dust2' and 'de_inferno' (the in-game name for the maps Dust 2 and Inferno).

- Numeric data, for any data point that is just a simple number.
- Boolean data, such as the target variable - Either the CT team wins or the T team wins, so that this variable can be converted to a boolean data type.
- Object data: This data type is the most versatile, as anything can be put in here, but models typically do not use this type and need to de-construct it into smaller pieces.

Picking the right data type to represent a value can be very important. Besides reducing the size of the data on memory, making the training and testing phase quicker, there are also special ways to deal with categorical and ordinal data that can make the final model much more effective at predicting outcomes, for example. This effect will be more explored in Section 3.2.2.3, where we talk about feature encoding.

3.2.2 Feature Engineering

With the data loaded and ready to work with, we then need to start extracting the needed features from it. Feature Engineering is the process of extracting more and more useful information from existing data [43]. The usage of domain knowledge in this process also tends to lead to better results.

3.2.2.1 Feature Transformation

One of the first and more straightforward things that can be done is to transform the data into a format that is more approachable by the models we want to train. One of the most common transformations is the standardisation of features that can be accomplished by removing the mean from it and then scaling the data to fit it for unit variance. This step is almost a requirement for many types of machine learning estimators because those estimators might not behave well if the data does not look similar to a normal distribution with a mean of 0 and unit variance. In the library scikit-learn, this transformation is already implemented and is very quickly used [5]. Other estimators seem to simply prefer a variable that has been scaled to be between 0 and 1, without any centring or changing the shape of the data. This is usually done in scikit-learn using the MinMaxScaler [6]. There are plenty of other ways that the data can be transformed when facing more specific issues, but in our work, the previously mentioned transformations were the ones that gave us the best results.

[5] sklearn StandardScaler
[6] sklearn MinMaxScaler

3.2.2.2 Feature Extraction

Feature extraction is a process of creating new features out of the information that we already have at our disposal. For this process, we had to take a very close look at the data set in order to see what could be extracted and what could be useful.

At first glance, there is considerable important information that we can extract with relative ease:

- The map that is being played on - it can completely change the way that the match is played. However, we ignored the version of the map or game, as CS:GO did not have any major updates that would significantly change the gameplay.

- Current score of both teams - This can be a significant indicator, as teams that are in the lead tend to be the best teams, and we assumed they would have a more significant chance of winning the round.

- Round Status and time left - We believe this to be an essential feature, as it can influence a lot the way players play: For example, if the bomb has been planted, it means that the roles have been inverted, and the terrorist team has to defend the objective (the bomb) until the time is up or the other team is dead, which can be an important factor. The time left in each phase of the round supplements this information; for example, if the Terrorist team is about to run out of time, they might be pressured to go on the offensive in order to plant the bomb or get a team wipe, as they do not have any more time to spare.

- Number of players alive in each team - If a team has been able to kill some of the enemy players during the round, they will have gained a numbers advantage, allowing for new strategies to be used, increasing their chances at a positive outcome. For this, we had to cycle through all the alive players and add them to their respective teams.

- And of course - which team ended up winning the round in the end, as this will be our target variable when creating our models.

With these basic features, we could have already started to develop the models, but there are many more interesting features that we can extract from the data:

- Team health levels - Sure, some teams can have five players alive, but if they are all at low health levels, it might be easier to be killed by surprise than if they were at full health.

- Equipment values in each team - The cumulative value of all the equipment that a team has at their disposal can be a strong indicator for who will win the round because a team that is better equipped and with better weapons tends to have an advantage during confrontations. For this, we had to compile a list of the values of all the possible equipment, from weapons to armour to the number of grenades each player has at their disposal at that moment.

- Money amounts per team - The equipment value is a piece of important information, however at the beginning of a match, during the buy phase or "FreezeTime", as expressed on the data set, players might still have not bought any equipment. However, the money that a team has on this phase sets an upper limit to how much can be bought for the team. While not as concrete as the equipment value, it can serve as an indication of what could be the equipment value in the near future.
- Do the Counter-Terrorists have a defuse kit with them? The answer to this question can make the difference in the outcome of a round where the Terrorists have planted the bomb: A person with a defuse kit can defuse a bomb in half of the time, and those few seconds can make the difference between a successful defuse and having the bomb exploding, making the Terrorist team win the round and possibly killing the remaining CT players.
- Total number of grenades and individual number of the different grenades types - Grenades have multiple different types and multiple purposes and uses. High-Explosive grenades can deal damage and even secure kills from afar without risking taking damage. Flashbangs can blind and deafen enemies and are perfect for breaching an area or stopping a player from being able to defend a particular angle, securing kills without the enemy being able to see the player. Molotov or Incendiary grenades leave an area on fire for a few seconds and are served as a severe deterrent from entering or passing over it, as it will deal severe damage to players, possibly even killing them if they stand for long enough, making them very powerful area-denial tools. Smoke grenades hinder the vision of players in the area, making movements more uncertain because of the lack of information on what is on the other side of the smoke cloud, and they can also extinguish an incendiary grenade instantly. As such, grenades have a critical role in the strategies that each team will employ. We believe that it was a good idea to collect the information on the number of grenades that each team has at their disposal, as it can make or break an encounter and ultimately the round.
- Bomb site where the bomb was planted - The reason for the extraction of this feature was because not all bomb sites have the same characteristics, so they have to be approached in different ways, with varying results. We believed this could be an interesting feature to explore due to this variability.
- Number of active smokes and incendiaries - The number of smoke clouds or areas of flames that are active in the game could mean that, for example, in conjunction with a low amount of time left on the clock that the defence is trying to block or hinder the entrances of a bomb site, and playing for time. For cases like these and others, we believed this feature would be an interesting addition to our models.
- If the game is in overtime - If a competitive match reaches the final round and the players are tied 15-15, the game can continue forward in overtime until one of the teams get a margin of two points to win. In this phase, some of the rules of the game change; for example, the money at the start of each round can be 10.000$ (ten thousand) or 16.000$ depending on the

version of the game, per player, making all players capable of buying almost anything they could want in a round. Since the rules of the economy change so drastically after this period, we believe that this feature could be fundamental to understanding the other variables in this phase.

- If the bomb was dropped - If the bomb is currently not in the inventory of any terrorist, it can indicate that the bomb is currently on the floor, potentially after the Terrorist player that had the bomb died, dropping it on the floor. This can be relevant because, in some situations, it might be beneficial for the Counter-Terrorists to defend a bomb that is on the floor instead of defending locations like the bomb sites, as the Terrorist team needs to get the bomb back if they ever want to plant it.

3.2.2.3 Feature Encoding

The features were all extracted, however, the data of some of them is not in good condition to be used. The best example of this is categorical data, where many of the columns of this data are either unusable or in a fashion that is not optimised for the different estimators. The features mentioned that have a categorical data type are the round winners (CT or T), map (8 different maps), round status (3 different possible round statuses - 'FreezeTime', 'Normal' and 'BombPlanted'), the bomb site of the bomb (either A or B but could be empty if the bomb has not been planted yet), if the game is in overtime (yes/no), if the bomb is on the ground (yes or no), and if the CT team has at least 1 defuse kit in their team (yes/no).

As we can see, all of these variables are of a nominal type, meaning there is no particular order to their categories, or categories that are "worth more" than others. Of the nominal features, there are even four of them that have two options, fitting into the Boolean type's description - winning team, if the bomb is dropped, if the CTs have a defuse kit, and if it is overtime.

So here is the problem: the estimators we use cannot understand text data, only numeric values. So we have to convert this text data in the form of categories into a more useful numerical format. This process is called feature encoding.

One of the simplest ways one can do this is called Integer Encoding, and to use it, we start by attributing a number to each possible category. Using the round status as an example, 'FreezeTime' could be converted to zero, 'Normal' to one, and 'BombPlanted' to two, making up the three different options. A similar encoding operation can be applied to the eight maps, going from 0 to 7, and to the four Boolean features, making them be 0 or 1. For ordinal features, this type of transformation is perfect if the order or priority of the categories is represented in the numbers. This simple encoding is also perfect for the boolean features, as it becomes a simple 0 and 1, perfect for most algorithms.

However, when it comes to nominal values, this encoding style runs into a bit of a problem: The estimators might assume there is a natural ordering to the categories when the relationship is entirely meaningless. This effect can deteriorate the performance of the final model and create unexpected results, so a different type of encoding is necessary. For nominal types, we primarily use a process called 'One-Hot Encoding', where we create a new binary feature for each category, setting only the correct binary feature to one, leaving all others at zero. In the case of the round status variable, it would result in three new variables being created, one for each of the three categories, and setting the binary features to one only in the respective variables. Also, it is important to note that these types of binary variables are also called "dummy variables" by some.

There is still something else that can be done to improve our feature encoding. Since One-Hot Encoding creates only one variable that is set to "1" in its categories, the effect of this is that we can predict with 100% accuracy at least one of the variables. This happens because, for example, if there are three categories and One-Hot encoding is applied, and if the two first variables are set to "0", then we can confidently say that the third and final variable will have "1" as its value. This effect where we can predict the value of a variable considering the values of other columns is called multicollinearity. In this case, since we can perfectly predict the values, we have what is called *Perfect Multicollinearity*.

The existence of multicollinearity can be a huge problem, especially for some regression models[11]. Linear Regression, for example, assumes that multicollinearity does not exist in the data set we provide and may simply not work when perfect multicollinearity is present. Besides that, the model would have to learn using more variables, which increases the computational power and time needed to train and test the model and also makes the resulting model more complex and difficult to understand. This effect can be caused, as we have seen, by the "dummy variables", creating the commonly called dummy variable trap - the creation of perfect multicollinearity by encoding categories into dummy variables.

In order to combat this issue, we can drop one of the variables created by this encoding style. This way, when all variables are "0", then we know that the variable we dropped was the missing "1", and no information is truly lost. While we will be using a classification model to predict the outcome of the round and not a regression model, cutting the extra variables in this model will simplify the model and reduce the computational power requirements while "keeping" the same information.

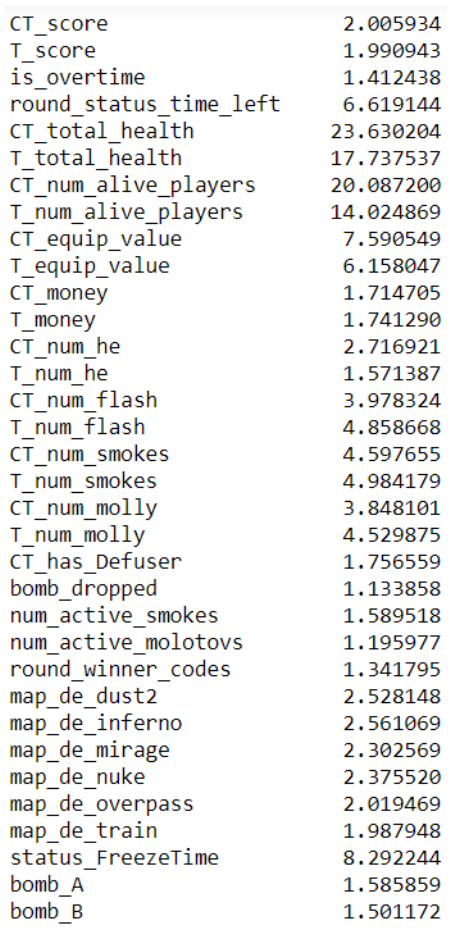
In our implementation, to avoid this issue, we drop one of the variables from the three categorical variables: We drop one of the maps, one of the round status variables, and one of the bomb sites (we chose the "null" variable, from when the bomb is not planted yet). With this, we are able to avoid some of the perfect multicollinearity that appeared through the encoding of these three categorical features.

3.2.2.4 Cleaning Related Features

At this point, we have already done quite a bit of feature extraction and transformation, but we may have unknowingly extracted information that can be duplicated, highly related or simply useless to our model. If two features have the same information, we do not need both of them, as they will not help our models with the classification process and might just add noise to the problem. With this in mind, we looked more closely at what could be done to detect and combat these situations.

One of the things that we can look into is something we mentioned in the previous section (3.2.2.3), multicollinearity. Multicollinearity is usually measured using the Variance Inflation Factor (VIF)[51], although some literature also mentions the correlation matrix.

We have calculated the VIF values for all our features, but when the results came in, we detected a curious situation: One of the encoded categories for the round status - the status where it demonstrated that a bomb had been planted - and the two remaining bomb site features (for bombs planted in A and B) displayed perfect multicollinearity since the VIF was infinite. After taking a closer look, it entirely made sense; when the bomb site was set to "A" or "B" and not empty, it meant that the bomb was planted, so the round status passed from "Normal" to "BombPlanted". When the bomb site and the round status variables got encoded, both "bomb in A" and "bomb in B" became their own separate features, and the same happened with "status BombPlanted". This meant that when either "Bomb A" and "Bomb B" were true, the status showed that the bomb was planted, and when neither bomb site was set to true, it meant that the bomb was not planted yet. This is a perfect multicollinearity, as we can predict one of these three features knowing the other two. This meant that we could drop one of these features and not lose information, simplifying the final model once more. While we had not realised this at first, this demonstrated that the data set has some redundant data, and our usage of it managed to slip through our work all the way here.



```
CT_score                   2.005934
T_score                    1.990943
is_overtime                1.412438
round_status_time_left     6.619144
CT_total_health           23.630204
T_total_health            17.737537
CT_num_alive_players      20.087200
T_num_alive_players       14.024869
CT_equip_value             7.590549
T_equip_value              6.158047
CT_money                   1.714705
T_money                    1.741290
CT_num_he                  2.716921
T_num_he                   1.571387
CT_num_flash               3.978324
T_num_flash                4.858668
CT_num_smokes              4.597655
T_num_smokes               4.984179
CT_num_molly               3.848101
T_num_molly                4.529875
CT_has_Defuser             1.756559
bomb_dropped               1.133858
num_active_smokes          1.589518
num_active_molotovs        1.195977
round_winner_codes         1.341795
map_de_dust2               2.528148
map_de_inferno             2.561069
map_de_mirage              2.302569
map_de_nuke                2.375520
map_de_overpass            2.019469
map_de_train               1.987948
status_FreezeTime          8.292244
bomb_A                     1.585859
bomb_B                     1.501172
```

Figure 3.2: Multicolinearity Values

After solving this issue, we noticed another perfect multicollinearity, this time related to the grenade numbers, but it was very straightforward. The problem was that we had, as features, both the number of individual grenade types bought by each team and the total number of grenades bought by each team. Since the total number is a direct sum of all individual types, another perfect multicollinearity was detected, and so we chose to drop the total number of grenades feature since we believed that the individual types were easier to understand and without any direct relation to the others.

Afterwards, we saw that the total health of each team and the number of alive players of each team were also displaying high VIF values (around 20 where most features had values around 1 or 2), meaning that these four features were nearly linearly related. This fact makes some sense since the number of alive players in a team is linked to each team's amount of health points. For example, when the round is still in the buy phase, there are five players alive in each phase that all have full health, so all those values naturally display very high correlations. Also, it is not uncommon that when a confrontation between two players starts, one might kill the other (with a headshot, for example) without suffering any damage, maintaining the players that are alive with maximum health. With this in mind, we can now understand how these values are highly related, although we believed that despite this, it would still be important to have both mostly for when the values are different, meaning that although many players are alive, some of them are injured, meaning they are much easier to down when a confrontation starts.

We can also look into the correlation matrix to see what information we can extract.

As we expected from what we saw previously with the multicollinearity test, There is an extremely high correlation between the number of alive players and the sum of the health of a team, but there is also a considerable amount of correlation between these variables and the ones in the opposite team. This is reasonable and expected considering the typical way a round develops. In most of the initial samples of a round, all players of both teams will be alive, and as the round advances and people start dying, it is very common to have the same number of players alive on both teams, even down to a final dual of 1 vs 1.

Another big area of positive correlation is in the relationship between the equipment value of each team and the number of grenades in each team's inventory. This was to be expected for two reasons: The grenades that were bought contribute to the total equipment value, so the more grenades a team has, the larger the equipment value tends to be, but also the fact that the more money a team has to spend on equipment, the easier it is to buy more grenades, as they can provide tactical options that a weapon improvement might not be able to provide. It is, of course, dependant on the personal preference of the players on how to budget their plays, but this effect needs to be still mentioned. The same sort of rationale can be applied to justify the correlation between grenades. The more grenades of a single type a team has, then the more common it is for the team to buy grenades of other types. This clearly demonstrates that teams do not tend to specialise in the types of grenades they buy, and typically a team buys a whole range of different grenades. On the Counter-Terrorist team, the fact that the team has at least one defusing kit in their inventory also has a positive correlation with the equipment value and the grenade numbers,

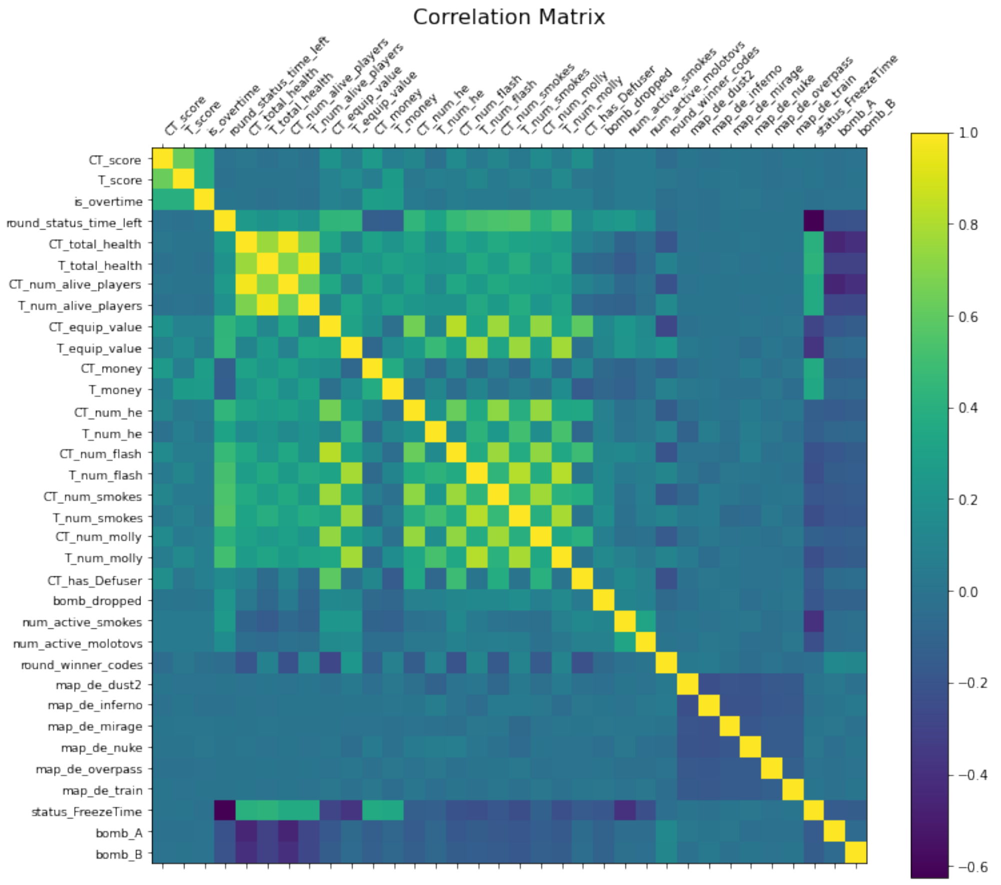


Figure 3.3: Correlation Matrix

and we suspect the reason for this is similar, as the more money a team has to spend, the easier it becomes to justify an individual purchase of this item.

The time left on this phase of a round (round_status_time_left) has also demonstrated some interesting correlations. There is a slight positive correlation with the equipment values of both teams, as well as the number of grenades available. We believe that this effect is due to two reasons. On one side, as the round progresses (and the time left gets smaller), the players will start to kill each other, and a dead player does not contribute its inventory for the equipment value or grenade numbers calculations. Also, as the round progresses, players will use the grenades they bought, so the number of grenades available also decreases. There is also a very negative correlation of the time left with the fact that a round is (1) or is not (0) in the "frozen" phase. This is perfectly normal since that phase only has 20 seconds, while the Normal phase has 115 seconds and the bomb timer has 40 seconds. This means that the lower values for the time left on the round are more common when the round is in the "frozen" state, making it have a negative correlation.

As we have seen from this correlation matrix, there are some correlations between the variables; however, there are multiple reasons to keep them in the model.

3.2.2.5 Feature Selection

We have seen in the last section some of the ways we used to detect some of the features that could be close to useless or even detrimental to our work; however, we may need to cull the number of features some more. If the data we currently have has many different features, we can run into some problems. One way to look at it is that the more features there are, the more "dimensions" the problem that is trying to be solved has, and thus the harder it is for the machine learning algorithms to group relevant data samples together, as the data set will become more sparse, reducing the statistical significance of the entire data. As such, it often becomes necessary an exponential increase in the data needed to build a model when the number of features or dimensions of the problem increase, or else the model might lose efficiency and become more sensitive to the effects of outliers and other such statistical phenomena. This effect is more commonly called the "Curse of Dimensionality", a term coined by R. Bellman [18]. This is something that we should avoid if possible. Besides, a smaller number of features can also make our model easier to understand once it is built so that we can extract some information from it.

With this goal in mind, we then proceeded to look into feature selection methods in order to select the best features for the model in an automated manner. There are different types of methods, and one of the distinctions that can be made is if these methods are supervised or unsupervised. When the outcome is ignored during the elimination of predictors, the technique is unsupervised. [33] Unsupervised feature selection techniques ignore the target variable, and examples of this are methods that remove redundant variables using correlation. On the other hand, supervised feature selection techniques use the target variable, such as methods that remove irrelevant variables.

There is also another important distinction to be made. The mechanisms used for feature selection in a supervised way can also be separated into wrapper or filter methods. Wrapper feature selection methods evaluate multiple different models with different subsets of input features in an attempt to find the optimal combination that maximises model performance according to a predetermined performance metric [33]. The model uses a predetermined algorithm, which is then wrapped around by these methods, hence the name wrapper methods. Filter methods, on the other hand, evaluate the relevance of each input variable to the target variable using statistical techniques, generating scores for each one in the process and then using these scores as a base for selecting the features that will be used in the model [33]. We also need to consider that some models can contain built-in or intrinsic feature selection, meaning that the model will only include variables that help maximise accuracy. In these cases, the model can pick and choose which representation of the data is best. Due to the very nature of how they operate, some algorithms are more resistant to non-informative variables. Tree- and rule-based models, such as MARS, Lasso and Random Forest, for example, intrinsically conduct feature selection [33].

In the previous section, we used some metrics such as the VIF and correlation matrix to explore and detect some of the potential issues of highly-correlated features. In some ways, since these metrics do not use the target variable, this can be seen as an unsupervised method for detecting and removing non-informative features, a sort of crude feature selection. However, we wanted to see if we would be able to identify some subsets of features that would improve the performance of the model.

In the wrapper methods area, we found that one method, called Recursive Feature Elimination (RFE), is very commonly used for this effect. The approach of this method involves the recursive removal of features and then building a model with the remaining features. It uses this model's accuracy to identify which combinations of features contribute the most to predicting the target variable. When the full model is created at first, the importance of each feature is computed, ranking them from most important to least. At each stage of the search, the least important features are iteratively eliminated prior to rebuilding the model [33], until the desired result or number of features is achieved. The importance of each feature is calculated either using a machine learning method that provides these scores, such as decision trees or by using statistical methods. The only thing that would need to be determined with RFE then becomes how many features do we want to keep, and this effect can be tuned either manually by methodical changing and testing, or using hyper-parameter tuning to find the values with the best results automatically.

In the filter methods area, the most basic way of doing things is by selecting a given number "k" of the features with the highest-ranking scores in a certain model. We can also, for example, choose a percentage of the features instead of a certain number. As for the way we can measure the relevance of each feature, we have a vast range of different options on what to use as statistical measures for the feature selection. We did notice that most of the statistical methods had very specific conditions on what kind of data the problem uses. Since our problem primarily uses numerical data as an input (and the categorical variables had their values encoded into numeric values) and uses a categorical value as an output or target variable (what team wins this round), then we need to look into what measures are adequate for this type of problem. One of the tried and tested metrics for this use is the Analysis of Variance (ANOVA) F-Values [44, 30]. The ANOVA method tests for (statistically significant) differences between multiple groups of data, in this case, multiple variables, and also how much does the data inside of a group/feature varies. If there is a large amount of variance between the features and within each feature, then there is evidence that we are looking at two different "populations". This results in the f-value being larger, which allows for the ranking of features by the level of importance. We can use this ranking to select the best features to build a model around. The number of features that we want to select, as with the RFE method, can be chosen and tested manually, or we can use hyper-parameter tuning to search for the best configuration automatically.

In our case, we tested both the Selection with the ANOVA F-values method and the RFE method, giving us similar results in our initial tests, and according to some sources [12], RFE tends to give better results, so we decided to choose this as the primary driver of feature selection.

3.2.3 Class Balancing

Before we start creating models and extracting information from them, there are still a few things that we need to check. Extracting some simple statistics from our data set can already provide some interesting information.

One of the things that we need to look into right at the start is the balance between the classes of the target variable that we use. If our data set demonstrates a significant difference in the numbers of samples of each class, then we will have a problem, since a skewed or biased data set can lead to overfitting on the most common classes, bad accuracy on the least common classes, and a general decrease in the real-world performance of a model. Most machine learning algorithms used for classification were designed with the assumption that an equal number of samples per class exists.

Another example of what can happen is that we fall into the commonly called Accuracy Paradox [53], where we can find that accuracy may not be a good metric for predictive classification models. An example of this happens when we have a class A being found in nearly all of the cases, so if the model just predicts that the result is always A without looking into anything else, the accuracy will be nearly perfect, but this model completely fails to predict the other classes and is almost useless in any real-world scenario. In this case, precision, recall and F1-Scores are better metrics to use, as they allow us to see the performance of the other less represented classes.

If this issue of class imbalance is detected in our work, we have a few options on what we can do to combat it. For example, if we were able to get more data into our analysis, maybe the new data set would show a different and perhaps more balanced distribution of the classes.

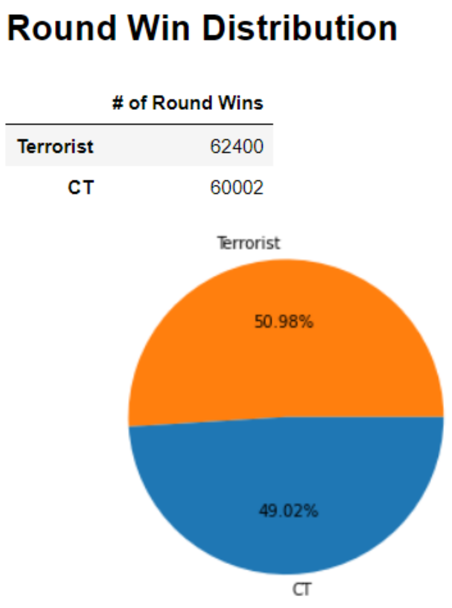
Another approach is to add copies of samples of the less represented classes (this is called over-sampling) or remove samples from the most represented classes (called under-sampling). If we have a large number of samples, we can try to under-sampling more; however, if the amount of samples from our data set is not very large, we might have to use over-sampling more. This approach is relatively simple to implement too, and can provide a boost to the true predictive abilities of the model.

If possible, it can also be worth considering the generation of samples from the less common classes in an artificial way. It works in a similar fashion to over-sampling but by creating new samples and not just copying the existing ones. There are many different methods of doing this: Kubat et al. demonstrated a method for under-sampling using a heuristic that balanced the data set through the elimination of redundant examples and noise in the majority class [32]. Nitesh et al. presented the SMOTE (Synthetic Minority Over-sampling Technique) method, which generated new synthetic samples by picking intermediate values between minority class samples and their nearest neighbour in order to over-sample the minority class [25]. Using methods such as the ones presented can help correct a class imbalance issue in a given data set.

3.2.3.1 Target Variable Distribution

For our use case, we then need to look into the balance between the classes of the target variable that we have. In our case, we need to check if both the teams have an approximate number of wins.

As we can see from the figure 3.4, we have a near-perfect distribution of the target variable, meaning it was almost as likely to win in either team. We see that there are slightly more Terrorist wins than Counter-Terrorist ones, but with the difference being nearly 1%, it is not significant.



Round Win Distribution

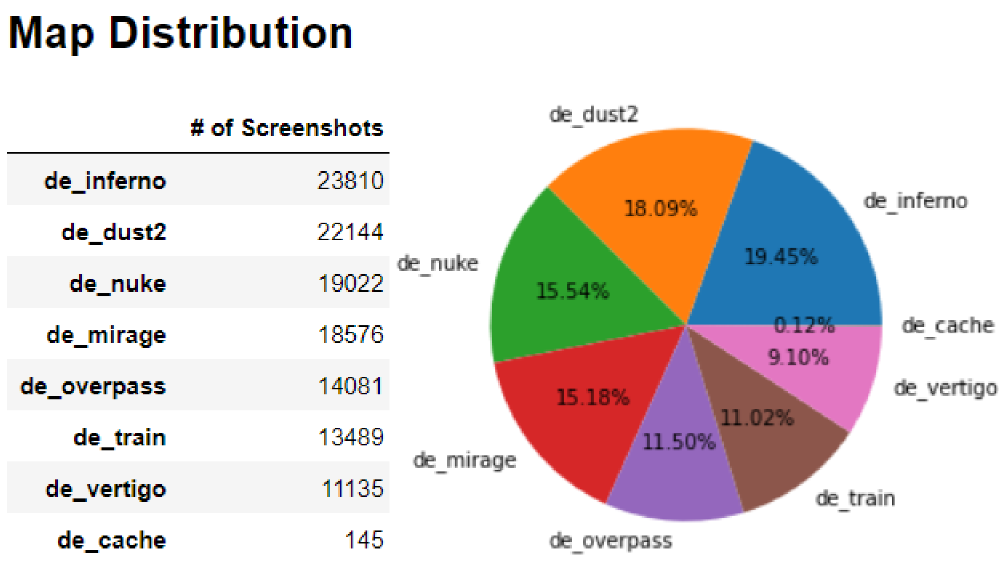
	# of Round Wins
Terrorist	62400
CT	60002

Figure 3.4: Round Win Distribution

Since our classes are well balanced, there was no need to use the previously mentioned methods, but this process is still part of the methodology we followed.

3.2.3.2 Map Category Distribution

Something else we can check is also the number of samples on each of the maps. The figure 3.5 displays a pie chart of the map distribution and their absolute number of samples.



Map Distribution

	# of Screenshots
de_inferno	23810
de_dust2	22144
de_nuke	19022
de_mirage	18576
de_overpass	14081
de_train	13489
de_vertigo	11135
de_cache	145

Figure 3.5: Map Distribution

Here we can notice a minor issue: the map de_cache has an extremely small representation in this data set. In fact, only 145 samples, while the others all have between 11 and 24 thousand samples each. Also, it is very possible that those 145 samples were all taken from one singular game, so the data from that map might be significantly influenced by the skill of the players involved in that one particular match. As such, we believed that the de_cache map, due to it being insignificant in the presence of the other maps and without any way of getting more samples to compensate, we decided to remove the 145 samples of this map from the model, making sure that we did not trip on the dummy variable trap from the Feature Encoding section (3.2.2.3) once more. The rest of the maps may be a bit imbalanced, but the fact that they have a substantial amount of samples each, and since this is not the target variable, we decide to leave it as is, as it should not create considerable discrepancies. Nevertheless, just in case, we did check for the relation of each class with the win rates of each team in a few cases, but the results of that will be explored in the Results section (4) of this thesis.

3.2.4 ML Algorithms and Pipelining

We performed several experiments with various different Machine Learning algorithms, specifically: Logistic Regression (LR), Decision Tree (DT), Random Forest(RF), Support Vector Machines (SVM) using a radial basis function as its kernel type, Linear Support Vector Machines (LSVM), Gradient Descent (SGD), Naive Bayes (NB), and Gradient Boosting Classifier (GBC). We make use of Scikit-Learn's implementations of these algorithms and mostly keep using the default parameter values.

In order to better evaluate the effectiveness of the created models, we use cross-validation, using a 5-fold stratified split of the data. The data in this split was also previously shuffled, so a better proportion of different matches was found in each fold. Without the shuffle, we found that many samples from (presumably) the same match were getting lumped together in the same folds, and so the models tended to have a very skewed basis for training and testing, degrading performance significantly. For each fold, we return the classification report implementation on scikit-learn, which calculates the accuracy, precision, recall, f1-score and support for each class. Although we calculate and report all of those, the one we care most about is accuracy, which gives us an idea of how well a certain model is performing. If our classes were severely imbalanced, then a better metric could be the f1-score, such as the macro-averaged one. In our case, since the classes are well balanced, this is not an issue.

When working with our models, we need to be careful in order to avoid creating data leakage. Data leakage can occur when information from the test data is being used to fit the model or parts of the model. An example of this would be if the scaler we used were going to be fit using the whole data set (test set included) and not just the training set. This means that information about the data in the test set was used to fit the model and can lead to misleading results.

In order to avoid the issue of data leakage and lower the risk of bugs in our implementation of all these steps, we thought it was wise to make our Pipeline be able to handle all of these transformations in a safe way that does not create data leakage, even when using cross-validation.

The scikit-learn has an existing implementation of a pipeline that is very easy to understand and work with, so we made quick use of it.

We want to look into what factors are the most important to the outcome of a round, and if we are able to detect that a particular data point is way more or less important than intended by the game designers, it could be an interesting alert that something might be wrong with the balancing of what affects those data points.

With this plan in mind, we want to perform some meta-analysis of the models to know what features were the ones that contributed the most to the results we obtained from the models we are preparing. To get this information, we can extract the feature importance values from the trained models.

3.3 Summary

In this chapter, we presented the general methodology that was utilised for this research work, along with the rationale between our decisions. We have described the data set in detail along with the way we processed it into more useful data, created machine learning models, evaluated the created models (and performance metrics that were used), and how we extracted information directly from the data set and also the models. In the next chapter, we will present the results we obtained from our approach and discuss their implications. The implementation that we have done can be seen in more detail in our GitHub repository [7].

Chapter 4

Results

Contents

This chapter shares, evaluates, and discusses the results obtained in this project for the proposed approach. We extracted some interesting statistics directly from the data set that we will demonstrate in section 4.1, and furthermore, we explore both the models and the features that lead the models created to consider which team would win a round in section 4.3. The code for this implementation of our use-case is publicly available in our GitHub repository [7].

4.1 Map Balance

One of the most defining characteristics of a match is the map on which the match will be played. Every single map has a unique layout, heavily influencing the ways players approach each other and how they attack and defend. For example, if a map has strong defensible areas on the bombsites, then it might make it easier for the Counter-Terrorist team to hold a bombsite and win rounds, or for the Terrorists to defend a bomb site if an attack is successful and the bomb planted there.

We compiled simple charts to better visualize this data in figure 4.1, and we can detect some inconsistencies. While the average win proportion for all maps in this data set is very slightly in favor of the Terrorist team (51% T / 49% CT), some maps demonstrate a clear leaning towards one team or the other.

The maps "Dust 2" and "Inferno", the two most played maps in this data set (figure 3.5), both have an increase in the incidence of Terrorist wins. On the other hand, the maps "Nuke" and "Train" have the opposite effect, with the Counter-Terrorist team being the ones with the most wins.

Besides the map win percentages, we also thought it would be interesting to look into the round win rates after the bomb has been planted. This way, we can look into how hard are the

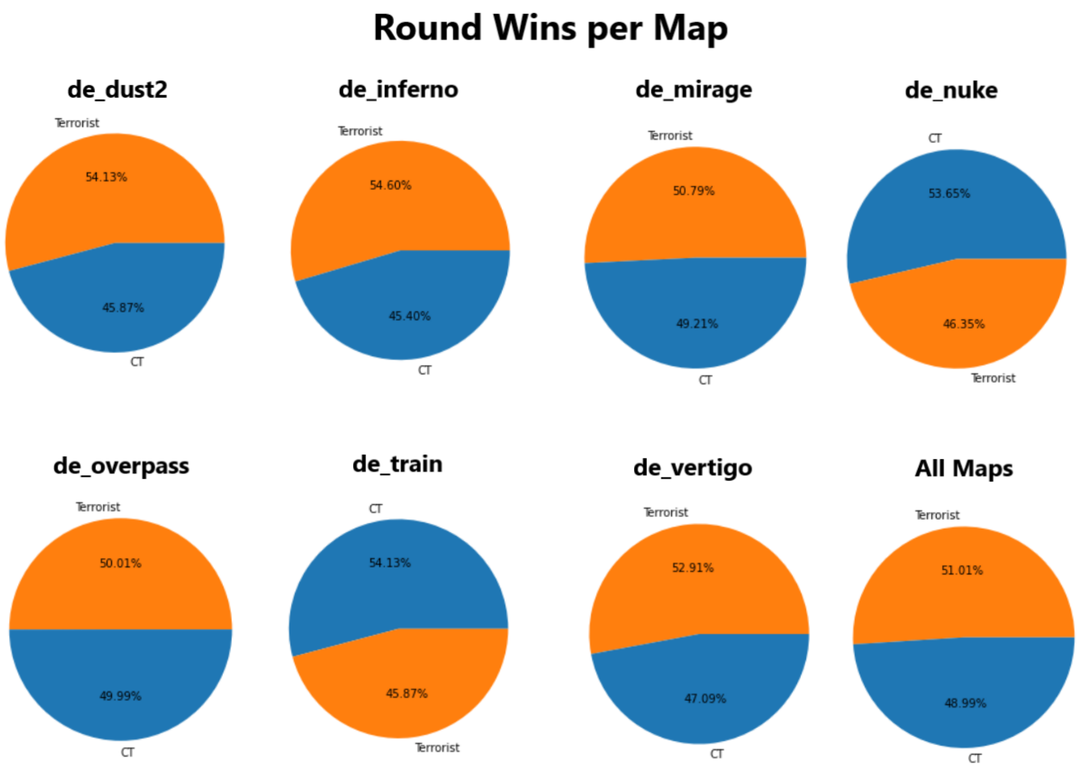
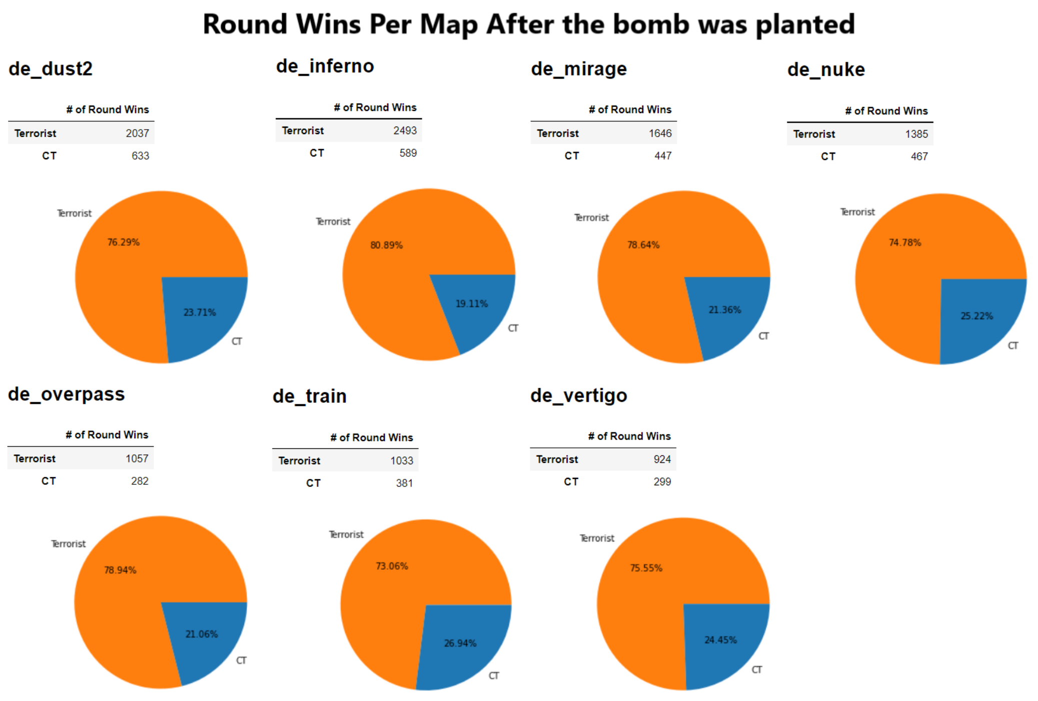


Figure 4.1: Round Wins per Map Distribution

bombsites of a specific map to be taken back by the Counter-Terrorists in time to defuse the bomb. Figure 4.2 presents the results of this collection, and there are a few things of note.

Some maps appear to have a low or high percentage of CT wins when compared to the other maps. The lowest maps are Inferno (with the lowest of 19.11% CT Wins), Mirage and Overpass. The maps with the higher CT win percentage are Train (with the highest of 26.94%), Nuke and Vertigo.

It is interesting to note that the maps where the CT team has higher chances of winning (i.e. Train and Nuke) are also the same ones where the CT team appears to have a higher percentage of successful bomb defuses. We could argue that this increase in CT wins after a bomb is planted is what makes the maps more CT-sided (meaning that the CT team has a higher chance of winning). While we agree that this may be a contributing factor, it cannot be the sole reason for it because the number of bomb defuses is not enough to account for the difference, and besides this, the map Dust 2 has an average bomb defuse rate, while at the same time being one of the most T-sided maps according to our statistics, and the other T-sided map Inferno has a low bomb defuse rate. This difference between the Dust 2 and Inferno map is curious, and it might show that one of the causes why the Inferno map is more T-sided can be related to the bomb and bombsites. The reasons for this increase in the Terrorist win rates and the decrease in defuse rates can be multiple and varied:



de_dust2

	# of Round Wins
Terrorist	2037
CT	633

de_inferno

	# of Round Wins
Terrorist	2493
CT	589

de_mirage

	# of Round Wins
Terrorist	1646
CT	447

de_nuke

	# of Round Wins
Terrorist	1385
CT	467

de_overpass

	# of Round Wins
Terrorist	1057
CT	282

de_train

	# of Round Wins
Terrorist	1033
CT	381

de_vertigo

	# of Round Wins
Terrorist	924
CT	299

Figure 4.2: Round Wins per Map after the bomb was planted

- The bombsites are easy to attack for the Terrorist team
  hard to defend by the CT players;
- The bombsites routes may be longer, meaning it takes more time for players to traverse the map to help teammates;
- The bombsites or the routes to them can be easily defended from one side and not the other

4.2 Equipment Value Difference

We also believed that equipment value would be a very important predictor of what the result of a round would be for a few different reasons:

- A team that is better equipped tends to have an advantage, as it might be able to both sustain more damage and even deal more damage to opposing players, and so, in direct confrontations would, in general, come out on top.
- Teams that have won more, especially in the latest rounds, tend to have a much better amount of funds to spend on equipment. This is because of the way the funds for equipment are won in this game: The winning team tends to get more funds for a win than it does for a loss, as players are awarded more money when they win than when they lose but are also awarded more money when they kill an enemy player. Besides this, players that are alive by the end of the round keep their equipment for when the next round starts, potentially saving the

money they would spend to buy new equipment. However, there are exceptions to this: if a team starts losing too many times in a row, the amount of money they receive will increase, potentially receiving more money than the amount that is received by the winning team.

With these reasons in mind, we wanted to explore more the relationship between the equipment values of each team and which team would win the round. For that, figure 4.3 was created by computing the equipment values of both teams in all samples and painting the graph according to what team won in the nearest samples to those coordinates, linearly interpolated. Counter-Terrorist wins are painted in blue, while Terrorist wins are painted yellow.

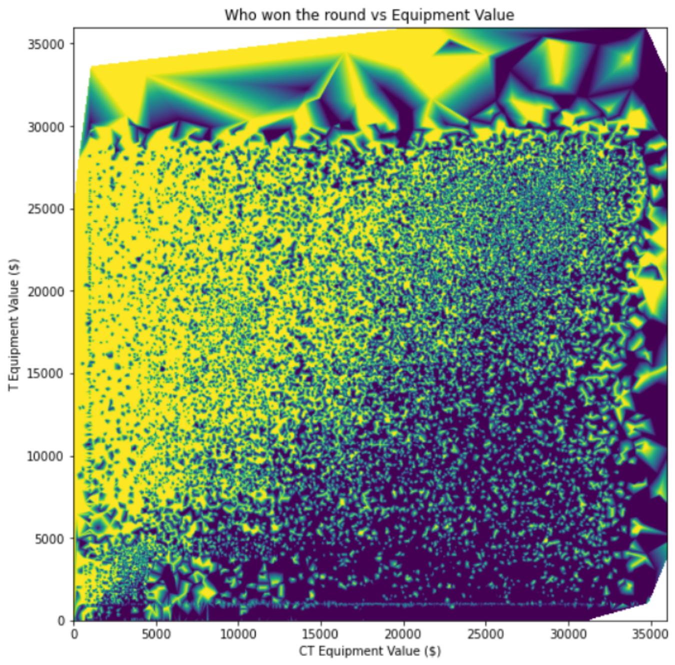


Figure 4.3: Equipment Value Difference influence on Outcomes

In this image, it is evident that when a team gets an equipment value that is much higher than the opposite team, that same team has a solid chance of winning the round. While we are able to see this apparent correlation between the equipment value and the chances of winning, there are a few areas of this figure that can raise a cause for concern.

One such area is that the average win change for the Terrorist team seems to severely drop to a "mixed" or noisy area (area between the red lines in figure 4.4) of the image near the 20000$ mark for CT Equipment as opposed to a much more clear divide in who wins in the lower values. Similarly, after the 16000$ price point for the T team equipment, the chances of winning for the T team seem not to increase as much as with previous values, possibly indicating that after that point, there is a reduction in the "expected return" for the money invested, meaning that spending more money than that won't increase the chances of winning, and might be wiser to save the money for the next rounds. This drop in the expected return is not as pronounced on the CT side since the region that is mostly blue continues very close to the diagonal line that is observed in lower values.

There are also some areas with unexpected densities for the colours of the opposite team, considering the difference in equipment values of both teams. One of such areas, highlighted in purple in figure 4.4, is the 8000$ to 12000$ in CT equipment values and 14000$ to 20000$ in the T equipment values. This area appears to have a higher density of CT wins than when compared to higher CT values or lower T values, as if there is a local optimum for the CT team, meaning that for them, it is more worthwhile to spend between 8000$ and 12000$ in equipment than to spend slightly more for better gear. This is strange and should probably be investigated as to what combination of purchases are leading to these equipment values, as there might be some specific combinations of weapons or equipment that are more powerful than more expensive alternatives, a clear sign that the cost of those items might not be balanced.

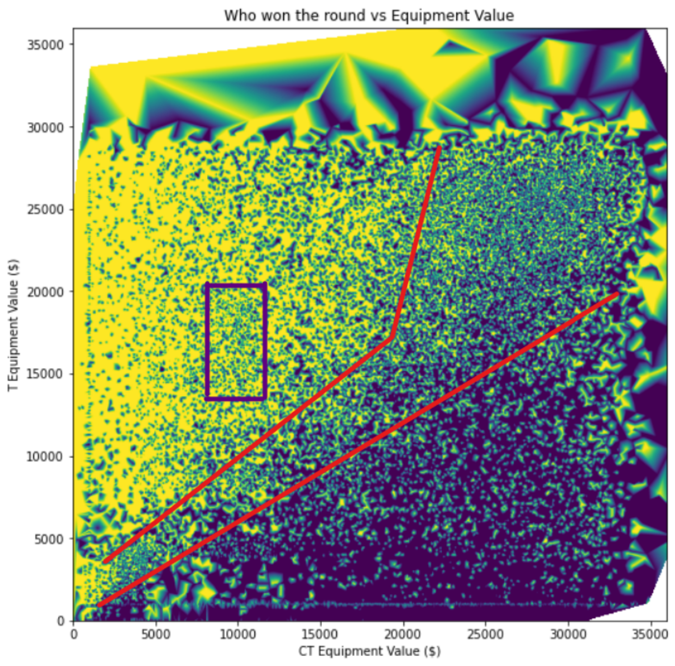


Figure 4.4: Figure 4.3 with regions of interest highlighted

4.3 Trained Models

For our experiments with the models, we attempt to predict the winning team without any regard to what stage the round is on, meaning that data from the start, middle and end of the rounds are used for this model. It is expected that for the final moments of a round, it should be slightly easier to predict the winning team, considering that at that point, teams usually have had multiple kills, and it is common that some teams have more members alive than others. This made us consider creating two different models, one for just the final stretch of the round and another for the beginning and middle of the round, where the round is still more undecided. However, we decided against this path of action for the risk of introducing selection bias on the model because since each round plays differently, we would not be able to undoubtedly identify what makes a data

Model	Accuracy	Weighted Average			Macro Average		
		Precision	Recall	F1-Score	Precision	Recall	F1-Score
Naive Bayes	0.69	0.69	0.69	0.69	0.69	0.69	0.69
Logistic Regression	0.71	0.71	0.71	0.71	0.71	0.71	0.71
Decision Tree	0.73	0.73	0.73	0.73	0.73	0.73	0.73
Random Forest	**0.78**	**0.78**	**0.78**	**0.78**	**0.78**	**0.78**	**0.78**
Gradient Boosting	0.71	0.72	0.71	0.71	0.72	0.71	0.71
SVC	0.72	0.72	0.72	0.72	0.72	0.72	0.72
Linear SVM	0.71	0.71	0.71	0.71	0.71	0.71	0.71
SGD	0.70	0.70	0.70	0.70	0.70	0.70	0.70

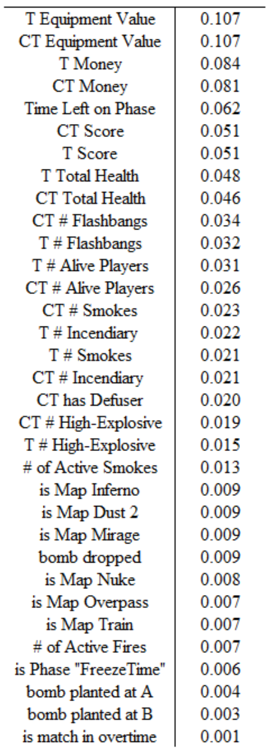
Table 4.1: Average results from 5-fold stratified cross-validation

sample good for one of those models or the other. Therefore, we thought that an all-encompassing model would be the best course of action.

We have obtained the results presented in Table 4.1. The table shows the average results from the 5-fold cross-validation setup in a binary (Terrorist vs CT) classification problem. The models do not present a very high accuracy, which makes sense given the unpredictable nature of the problem. The Naive Bayes and the Gradient Descent algorithms had the worst performances out of all the presented models, while the Random Forest algorithm has presented the best scores across the board by a comfortable margin, so this will be the model that we will be discussing about primarily.

With this model fit, we can then proceed with a meta-analysis of the model, and for that objective, we are then able to extract the feature importance values for each of the features we presented to this model. It is important to note that, since the Random Forest (and Decision Tree) algorithms make intrinsic feature selection, we have not performed any feature selection method on the feature and data that were used to this model. The feature importance values that we were able to extract are presented in Table 4.2.

We can see clearly from the feature importance values that the most important features are the equipment values for each team and the money each team has at the beginning of a round, followed by the time left for the round to end or change phase, as well as the scores of both teams up to that point. The total health points of both teams also follow behind but are considered more important than the actual number of players for either team. The number of flashbangs for the players of each team was considered a more important factor than any other grenade type, and with this information, we can interpret it like this item may be very strong, potentially for its price point or in its effects, so reducing the blinding effects of the grenade or increasing its price could bring this grenade more in line with the others. The opposite effect happened with the High Explosive grenade, meaning that if a team has none or many, it does not tend to influence much the result, so either this grenade could have its price dropped or have its impact affect a larger area. The map this match is being played on does not seem to be a very important factor in the decision, nor where in the map has the bomb been placed.



T Equipment Value	0.107
CT Equipment Value	0.107
T Money	0.084
CT Money	0.081
Time Left on Phase	0.062
CT Score	0.051
T Score	0.051
T Total Health	0.048
CT Total Health	0.046
CT # Flashbangs	0.034
T # Flashbangs	0.032
T # Alive Players	0.031
CT # Alive Players	0.026
CT # Smokes	0.023
T # Incendiary	0.022
T # Smokes	0.021
CT # Incendiary	0.021
CT has Defuser	0.020
CT # High-Explosive	0.019
T # High-Explosive	0.015
# of Active Smokes	0.013
is Map Inferno	0.009
is Map Dust 2	0.009
is Map Mirage	0.009
bomb dropped	0.009
is Map Nuke	0.008
is Map Overpass	0.007
is Map Train	0.007
# of Active Fires	0.007
is Phase "FreezeTime"	0.006
bomb planted at A	0.004
bomb planted at B	0.003
is match in overtime	0.001

Table 4.2: Feature Importance values from Random Forest algorithm

Chapter 5

Conclusions

Contents

In the competitive games scene, balance is often a very tricky topic to tackle. Maintaining all the options at the intended power levels is an arduous process, and it can hurt the players' enjoyment of the game if poorly executed.

With this project, we have gained more insight into game balancing in competitive games and the application of machine learning as a tool for this area. We have used classification algorithms to determine which team was going to win a round and deconstructed the model to learn more about how it makes decisions, giving us some insight into what makes players more likely to win.

This data-driven approach can help with the balancing process of a game, detecting balancing issues faster, reducing the necessary work for correcting issues.

5.1 Limitations

One of the limitations of this study is the fact that the data set we use has some data missing that could be very helpful for the algorithm. Data such as the identification of the players that are alive, along with metrics for their skill levels, could have improved the accuracy of the models. This data is very easily attainable if the company that developed the game wants to, as metrics like these are usually implemented for the typical matchmaking systems. CS:GO implements a rank based matchmaking system, so this information could already be helpful for our models if we had it available. However, the lack of this information, as previously described in section 3.1.1.1 can also be helpful in establishing a more generic baseline for competitive games and not be so specific for this one particular game.

5.2 Results

The presented dissertation proposed a classification pipeline for the process of identifying possible balancing issues in competitive games. Three research objectives were formulated to guide our research work and our methodology:

- Research common ML techniques for game balancing;
- Design a methodology that fosters supervised learning to help detect possible balancing issues on competitive games;
- Implement a case study based on a previously developed competitive game;

This dissertation has the aim to fulfil these research objectives. We have accomplished them with varying degrees of success, and we will break them down into more details for each one of them:

- Research common ML techniques for game balancing;

This research objective was achieved by our literature review in chapter 2, and even more in specific in the section 2.2.5, where we investigated the various different techniques that were used in the creation of mechanisms that would improve a game's balance for their players, both directly (as is the case of the AI Director from Left 4 Dead game, dictating various game mechanics) and indirectly (e.g. in the form of AI for fighting games that changes according to player skill levels). Furthermore, it was also observed that there are many different ways through which developers have attempted to tackle this issue, and their experiences were invaluable to our work.

- Design a methodology that fosters supervised learning to help detect possible balancing issues on competitive games;

This research objective was achieved by our proposal for a supervised learning pipeline that is able to characterize the importance of different values so that we can more easily detect, in a data-driven manner, what is kinds of actions can be possible sources of imbalance in competitive games. This pipeline, the description of how it works, and the rationale behind every step of its creation can be found in Chapter 3 and its various sections, describing all the major steps in the process.

- Implement a case study based on a previously developed competitive game;

We fulfilled this research objective with our case study using the game Counter-Strike: Global Offensive as a basis for our work. The implementation of this case study followed the methodology presented in chapter 3, and resulted in the creation of a classification pipeline that predicts what team will win the round given the current state of the game. The best model we created was able to predict the results with 78% accuracy. From this model, we then extracted some information about the way it worked that gave us some insight into some possible sources of imbalance in the game. While the results were not perfect or directly informative, a deeper investigation into what is causing the detected artefacts could lead to the very source of the issue and how to fix it.

For a more detailed observation of our use-case implementation, a GitHub repository with all the code is publicly available [7].

5.3 Future Work

For future work, we think that testing the robustness of this methodology is necessary, potentially with the usage of different competitive games. This task would be a big help in proving that this approach is not just a coincidence created by the chosen game and data set.

There are also a few factors that could be interesting to add to the model, some of which already discussed previously in the limitations (5.1), such as including information about the skill levels of the players involved, but include as well the current position of the players and of where the kills happen on the maps, in order to improve the model's accuracy and also detect hot-spots for where the kills and deaths happen.

Some new and variate data sources could also fill in some gaps in the knowledge of the models. One example of this could be the usage of CS:GO match demos (something similar to replays) to extract more detailed and specific data. There already exist some open-source projects that can be used to extract this data [4, 8], so this would not be an issue. However, the access to a relevant number of quality demos would create its own unique set of challenges.

Furthermore, a different project could be interesting: Since the place where the bomb was placed and the bomb variables were not considered to be very important, we could test these models versus data sets from other game modes, such as the Hostage Rescue mode, where the Counter-Terrorist team must try to extract a hostage from a map typically defended by the Terrorist team. By testing versus new modes, we try to confirm if some of the problems we have identified in the results are still present in other modes of play. We can also see if we can perform a satisfactory knowledge transfer from one game mode to the other. We considered the possibility to test this, however considering the objective of this research, we thought it would not be necessary and better suited for a follow-up study.

Our work could also be used to detect imbalances in a game and use that information as a basis for another work using an adaptivity-oriented approach. The results of our work could then be used to decide how to adapt the game mechanics to, for instance, maximise player engagement.

With the information that we also have, we believe we could have created a recommender system that would give advice as to where the player could go by having the knowledge of the areas where players most often kill or die, thus leading players into the areas that are perceived to be less dangerous. A system like this could be beneficial to players and teams to improve their chances of winning.

References

[1] Counter-strike: Global offensive » damage control.

[2] Cs:go ai challenge - copenhagen ai hackathon. https://web.archive.org/web/20200609053547/https://csgo.ai/welcome.

[3] Cs:go ai challenge - results. https://csgo.greatercph.com/.

[4] Csgo demos manager · akiver · github.

[5] Csgo.ai competition githhub. https://github.com/azohc/csai.

[6] The director.

[7] Egzyst/cs-go-ml-predictions - adaptivity in competitive games github repository.

[8] Github - markus-wa - demoinfocs-golang: High performance cs:go demo parser for go (demoinfo).

[9] In depth analysis of december 10, 2015 r8 revolver nerf (no other weapon changes included in update) : Globaloffensive.

[10] Senpai.gg - ai-based personal gaming assistant.

[11] The problem of multicollinearity. *Understanding Regression Analysis*, pages 176–180, 11 1997.

[12] Sulaiman Olaniyi Abdulsalam, Abubakar Adamu Mohammed, Jumoke Falilat Ajao, Ronke S. Babatunde, Roseline Oluwaseun Ogundokun, Chiebuka T. Nnodim, and Micheal Olaolu Arowolo. Performance evaluation of anova and rfe algorithms for classifying microarray dataset using svm. In Marinos Themistocleous, Maria Papadaki, and Muhammad Mustafa Kamal, editors, *Information Systems*, pages 480–492, Cham, 2020. Springer International Publishing.

[13] Ramiro A. Agis, Andrea Cohen, and Diego C. Martínez. Argumentative ai director using defeasible logic programming. *Communications in Computer and Information Science*, 614:96–111, 2015.

[14] Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. Automatic computer game balancing: A reinforcement learning approach. ACM Press, 2005.

[15] Mariette Awad and Rahul Khanna. Machine learning, 2015.

[16] Alexander Baldwin, Daniel Johnson, and Peta A. Wyeth. The effect of multiplayer dynamic difficulty adjustment on the player experience of video games. pages 1489–1494, 2014.

[17] Philipp Beau and Sander Bakkes. Automated game balancing of asymmetric video games. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 1–8, 2016.

[18] Richard Bellman. *Dynamic programming*. Princeton University Press, 1957.

[19] Michael Booth. The ai systems of left 4 dead.

[20] Jason Brownlee. How to perform data cleaning for machine learning with python, 3 2020.

[21] David Buckley, Ke Chen, and Joshua Knowles. Predicting skill from gameplay input to a first-person shooter. 2013.

[22] David Buckley, Ke Chen, and Joshua Knowles. Rapid skill capture in a first-person shooter. *IEEE Transactions on Computational Intelligence and AI in Games*, 9:63–75, 3 2017.

[23] Andriy Burkov. *The Hundred-Page Machine Learning Book*. 2019.

[24] Alex J Champandard. 11 secrets about left 4 dead's ai director and its procedural zombie director and its procedural zombie population. 2009.

[25] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[26] Felipe Machado Figueira, Lucas Nascimento, Jose da Silva Junior, Troy Kohwalter, Leonardo Murta, and Esteban Clua. Bing: A framework for dynamic game balancing using provenance. In *2018 17th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, pages 57–5709, 2018.

[27] Mark E Glickman and Albyn C Jones. Rating the chess rating system. *CHANCE-BERLIN THEN NEW YORK-*, 12:21–28, 1999.

[28] Carl Gutwin, Rodrigo Vicencio-Moreira, and Regan L. Mandryk. Does helping hurt? aiming assistance and skill development in a first-person shooter game. pages 338–349. ACM, 10 2016.

[29] Ralf Herbrich, Tom Minka, and Thore Graepel. Trueskill™: A bayesian skill rating system. In *Proceedings of the 19th international conference on neural information processing systems*, pages 569–576, 2006.

[30] David C Hoaglin, Frederick Mosteller, and John W Tukey. *Fundamentals of exploratory analysis of variance*, volume 261. John Wiley & Sons, 1991.

[31] Robin Hunicke. The case for dynamic difficulty adjustment in games. ACM Press, 2005.

[32] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *Icml*, volume 97, pages 179–186. Citeseer, 1997.

[33] Max Kuhn and Kjell Johnson. Applied predictive modeling, 2013.

[34] Matt Makes Games. Celeste, 2018.

[35] Tom Minka, Ryan Cleven, and Yordan Zaykov. Trueskill 2: An improved bayesian skill rating system. Technical Report MSR-TR-2018-8, Microsoft, March 2018.

[36] Jurike V. Moniaga, Andry Chowanda, Agus Prima, Oscar, and M. Dimas Tri Rizqi. Facial expression recognition as dynamic game balancing system. *Procedia Computer Science*, 135:361–368, 2018.

[37] Newzoo. Global games market report 2021.

[38] Nintendo. Mario Kart™ 8 Deluxe, 2017.

[39] Ashey Noblega, Aline Paes, and Esteban Clua. Towards adaptive deep reinforcement game balancing.

[40] Jacob Kaae Olesen, Georgios N. Yannakakis, and John Hallam. Real-time challenge balance in an rts game using rtneat. pages 87–94, 2008.

[41] Johannes Pfau, Antonios Liapis, Georg Volkmar, Georgios N. Yannakakis, and Rainer Malaka. Dungeons amp; replicants: Automated game balancing via deep player behavior modeling. In *2020 IEEE Conference on Games (CoG)*, pages 431–438, 2020.

[42] Geber Ramalho, Hugo Santana, and Vincent Corruble. Extending reinforcement learning to provide dynamic game balancing, 2005.

[43] Sunil Ray. A complete guide to data exploration. https://www.analyticsvidhya.com/blog/2016/01/guide-data-exploration/, 1 2016.

[44] Henry Scheffe. *The analysis of variance*, volume 72. John Wiley & Sons, 1999.

[45] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[46] Mariia Shakhova and Aleksandr Zagarskikh. Dynamic difficulty adjustment with a simplification ability using neuroevolution. volume 156, pages 395–403. Elsevier B.V., 1 2019.

[47] Rafet Sifa, Eric Pawlakos, Kevin Zhai, Sai Haran, Rohan Jha, Diego Klabjan, Anders Drachen, and Anders 2018 Drachen. Controlling the crucible: A novel pvp recommender systems framework for destiny. volume 10. ACM, 2018.

[48] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, 362(6419):1140–1144, 2018.

[49] Penelope Sweetser and Peta Wyeth. Gameflow. *Computers in Entertainment*, 3:3–3, 7 2005.

[50] Smishad Thomas. Data cleaning in machine learning: Best practices and methods, 2019.

[51] Christopher Glen Thompson, Rae Seon Kim, Ariel M. Aloe, and Betsy Jane Becker. Extracting the variance inflation factor and other multicollinearity diagnostics from typical regression results. *Basic and Applied Social Psychology*, 39:81–90, 3 2017.

[52] Simon Tripp, Martin Grueber, Joseph Simkins, and Dylan Yetter. Video games in the 21st century: The 2020 economic impact report.

[53] Francisco J Valverde-Albacete and Carmen Peláez-Moreno. 100% classification accuracy considered harmful: The normalized information transfer factor explains the accuracy paradox. *PloS one*, 9(1):e84217, 2014.

[54] Rodrigo Vicencio-Moreira, Regan L. Mandryk, and Carl Gutwin. Now you can compete with anyone: Balancing players of different skill levels in a first-person shooter game. volume 2015-April, pages 2255–2264. ACM Press, 2015.

[55] Rodrigo Vicencio-Moriera, Regan L. Mandryk, Carl Gutwin, and Scott Bateman. The effectiveness (or lack thereof) of aim-assist techniques in first-person shooter games. pages 937–946. Association for Computing Machinery, 4 2014.

[56] Christopher JCH Watkins and Peter Dayan. Q-learning. *Machine learning*, 8(3-4):279–292, 1992.

[57] Austin Wood. Evolve dev says '4v1 caused more problems than we ever imagined' | pc gamer, 6 2018.