4-18-2023 2:30 PM

IMPLEMENTATION OF A PRE-ASSESSMENT MODULE TO IMPROVE THE INITIAL PLAYER EXPERIENCE USING PREVIOUS GAMING INFORMATION

Rafael David Segistan Canizales, *Western University*

Supervisor: Mike Katchabaw, *The University of Western Ontario*

A thesis submitted in partial fulfillment of the requirements for the Master of Science degree in Computer Science

Abstract

The gaming industry has become one of the largest and most profitable industries today. According to market research, the industry revenues will pass $200 Billion and are expected to reach another $20 Billion in 2024. With the industry growing rapidly, players have become more demanding, expecting better content and quality. This means that game studios need new and innovative ways to make their games more enjoyable. One technique used to improve the player experience is DDA (Dynamic Difficulty Adjustment). It leverages the current player state to perform different adjustments during the game to tune the difficulty delivered to the player to be more in line with their expectations and capabilities. In this thesis, we will explore and test the ability to obtain the difficulty level in which a player should be placed initially, by using previous gaming information from platforms like Steam, combined with different machine learning (ML) algorithms and data analyses., In doing so, we can create a pre-assessment of the player as a way of improving DDA's initial state and the overall gaming experience of players.

Keywords

Video games, Artificial Intelligence, Game Analytics, Gamers Pre-assessment, Machine Learning, Dynamic Difficulty Adjustment.

Summary for Lay Audience

With the gaming industry growing rapidly, players expect better content and quality. One technique that is being used to improve the player experience is Dynamic Difficulty Adjustment (DDA). DDA systems use the current player data (Health, Score, Damage, etc.) to adjust the difficulty level of the game, making it more in line with their expectations and capabilities. This thesis explores how machine learning (ML) algorithms and data analysis can be used to obtain the initial difficulty level that a player should be placed at, using previous gaming information from platforms like Steam. This pre-assessment can improve DDA's initial adjustment and the overall gaming experience of players.

Dedication

I want to dedicate this thesis to my family, especially my wife, who has been a constant source of support, encouragement, and inspiration throughout this journey. Her unwavering love and belief in me have been instrumental in all of my accomplishments and goals. I am grateful for her presence in my life and I dedicate this work to her with all my heart.

Acknowledgment

First, I want to thank the SENACYT, Caldo, and Western University for allowing me to live such an incredible experience. Especially, I want to thank Bibiana, Silvana, and Adrian for all the amazing support throughout my studies.

I would like to express my heartfelt gratitude to my supervisor, Dr. Michael Katchabaw, for their invaluable guidance, support, and encouragement throughout my research journey. Their expertise and patience have been instrumental in helping me complete this thesis.

I also want to thank Hannan Lutfiyya, Anwar Haque and George Gadanidis for serving on my thesis committee and for their valuable feedback and suggestions. Their insights and critiques have greatly improved the quality of my work.

I am deeply grateful to all of my colleagues and friends who have supported me throughout this process. In particular, I want to thank Efrain Perez for their constant encouragement and for being a sounding board for my ideas.

Finally, I want to thank my family for their love and support, especially my wonderful wife Ambar Campble. Without their unwavering encouragement and belief in me, I would not have been able to complete this journey.

This research would not have been possible without the support of all of these individuals, and I am forever grateful for their contributions to my work.

Table of Content

List of Tables

List of Figures

Chapter 1 - Introduction

1.1 Motivation

The gaming industry has become one of the largest and most profitable industries of this era. According to market research company Newzoo's Global Games Market Report, video game industry revenues will pass $200 Billion dollars and are expected to reach another $20 Billion in 2024 (Wijman, 2022). With the industry growing as it is, players have become more demanding with the passing of the years, expecting better performance and great-quality games.

Game studios always try to find new and innovative ways of making games more enjoyable for players. However, since every player has different skill levels depending on the type of game, developers lack information when balancing difficulty curves. For flexibility, developers are increasingly looking to Dynamic Difficulty Adjustment (DDA), to tune various aspects of a game's difficulty to the particular needs and skills of individual players as they play. While this can work well, these approaches invariably have an initial adjustment period in which the play experience is suboptimal to the player, with the potential for the player churning or leaving out of boredom or frustration before the game can deliver a better experience. This is obviously a problem that needs to be addressed.

Researchers have used video games in the past few years to explore multiple machine-learning algorithms that perform different tasks. However, in comparison with other areas like Non-Player Characters' behaviors (Zhou et al., 2006) or Procedural Content Generation (Togelius et al., 2016), no study has explored the possibility of using previous player gaming data to improve the experience on a more personalized level.

Also, in Game Analytics, there are multiple studies using the player information stored in platforms like Steam, to predict purchase behavior in future games (Sifa et al., 2014; 2021) and even classify players based on their games and the time played on each of them (Zagreb, 2018). Many of these studies seek a similar objective, understanding and improving an aspect of the player experience.

In today's data-driven world, the potential of leveraging existing data on platforms like Valve-Steam or Microsoft-Xbox cannot be ignored. The present thesis endeavors to offer a fresh perspective on player skill classification by employing previous gaming data of players available on the Steam platform, with a specific focus on their achievements and games. Leveraging this data and using it within a DDA system presents exciting and potentially valuable opportunities for developers and players of their games.

1.2 Contributions

Our research aims to add a new layer to the DDAs that provides an accurate pre-assessment of the players. Doing so greatly increases the probability of DDA correctly assessing the player's ability from the beginning instead of waiting until enough information is gathered while the game is being played. This, in turn, results in a more optimal player experience earlier, before the player begins to contemplate churning. In this work, we want to explore the potential of using previous gaming information of players to predict which skill level they should be placed at, primarily based on achievements and games.

The concept of using a pre-assessment to improve the early performance of DDA systems is a novel one. To fully validate the potential of such a mechanism, one would need to integrate a DDA system into a previously unseen and unplayed game, add a pre-assessment module, and then

conduct user playtesting. Given the lack of suitable game candidates, the path forward would be to build a new game and proceed from there, which is an incredibly resource-intensive task. To justify this, we need more than mere intuition that pre-assessment would be beneficial. To that end, in this work, we conduct experimentation to verify the ability of past play experiences to predict future play outcomes in different but related games. Success in this regard would support the intended use of a pre-assessment module. (And, at the same time, provide the groundwork needed to build such a module down the road.) This is the primary goal of this work.

1.3 Roadmap

This thesis is organized as follows. Chapter 2 presents an overview of the core fundamentals utilized throughout this thesis to achieve our goal. In Chapter 3, we discuss the overall architecture and the work involved in creating our dataset of players. In Chapter 4, we apply the K-means unsupervised learning algorithm to label our newly generated dataset and conduct experiments using various supervised learning algorithms. In Chapter 5, we employ K-fold Cross-Validation to validate our supervised models and select the most optimal one. Finally, Chapter 6 offers a comprehensive analysis of our overall approach, contributions, limitations, and future directions.

Chapter 2 - Fundamentals

This chapter aims to provide a comprehensive overview of the fundamental concepts and principles that form the foundation of the research. The content of this chapter provides a theoretical framework for the subsequent chapters of the thesis and will serve as a reference for the reader throughout the rest of the document.

This chapter is comprised of three main parts. Firstly, we provide an overview of the fundamental concepts. Secondly, we discuss the related works and literature that have been published. Finally, we identify gaps in the current research space and position our own work in relation to these gaps.

2.1 Dynamic Difficulty Adjustment (DDA)

Dynamic Difficulty Adjustment (DDA) is a technique of programmatically modifying a game's options, behaviors, and scenarios in real-time, depending on the player's skill, so that the player, once the sport is straightforward, does not feel bored or annoyed when it's challenging (Zohaib, 2018).

The concept of DDA has been around for several decades, with early examples of its use in arcade games such as Pac-Man and Space Invaders. In recent years, with the advancement of technology and the rise of online gaming, DDA has become increasingly popular and sophisticated.

The purpose of the DDA is to keep the player engaged until the end and to provide them with a challenging experience. In standard games, game developers apply predetermined curves to control the level of difficulty as they go through trial and error, but constructing the curves is a time-consuming and complicated operation because of varying user demands (Moon et al., 2022).

There are several methods of implementing DDA in games. One approach is to adjust the game's parameters, such as the number of enemies or the speed of their movement, based on the player's performance. Another approach is to use machine learning algorithms to predict the player's skill level and adjust the difficulty accordingly.

Features like frequency, beginning levels, or rates can be set solely at the start of the game by selecting the level of difficulty. This can, however, lead to negative experience for players as they struggle to map a pre-established learning curve. DDA tries to resolve this drawback by presenting a customized solution for gamers.

Research has shown that accommodating the difficulty to the right spot can trigger a sensation of confidence, which increases motivation and the likeliness that the player will keep enjoying for an extended period (Constant & Levieux, 2019).

2.2 Game Analytics

Game analytics is the process of collecting, analyzing, and using data from video games to improve the overall gaming experience (Drachen et al., 2013). This data can include information on players' behavior, such as how they play the game, what they enjoy, and what they dislike. Game analytics can also include information on player engagement, monetization strategies, and game mechanics.

By analyzing this data, game developers can make data-driven decisions to improve the game. For example, they can use the data to identify areas of the game that need improvement, design effective monetization strategies, and keep players engaged. Game analytics has become an important tool for game developers, as it provides valuable insights into players' behavior and helps to ensure that games are optimized for player satisfaction. (Drachen et al., 2013)

Many studies have been performed to understand gamers on a higher level, using different statistical methods to group them and generate important decisions that impact game developers. From predicting the outcome of a certain game, or genre based purely on the reviews of other similar games, to classifying gamers by analyzing huge clusters of player data (Sifa et al., 2014, 2021), game analytics play a crucial role in helping game developers adapt to changing player needs and preferences, and create more engaging and profitable games.

2.3 Machine Learning

IBM defines Machine Learning (ML) as a branch of artificial intelligence (AI) and computer science that focuses on the use of data and algorithms to imitate the way that humans learn, gradually improving its accuracy (IBM Cloud Education, 2020).

2.3.1 Types of ML

Machine Learning programs can be categorized based on how much and what kind of supervision they require during training. The three primary types of learning are (Aurélien Géron, 2019):

2.3.1.1 Supervised Learning

Supervised learning is when we train the machine with well-labeled data. This indicates that some data has already been labeled with the right answer. Following that, the machine is given a fresh collection of examples (test data) so that the supervised learning algorithm may analyze the training data (set of training instances) and create a proper result from labeled data.

2.3.1.2 Unsupervised Learning:

This type of learning searches for patterns in a dataset with no labels. This form of machine learning, as the name implies, is unsupervised and requires little human supervision and preparation. Unsupervised learning is less biased than other kinds of AI since it does not rely on labels to discover patterns.

2.3.1.3 Reinforcement Learning:

Uses a learning system called an agent that interacts with an environment, selects and performs actions, and gets rewards in return (or penalties in the form of negative rewards). It must then learn by itself what the best strategy is, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

2.3.2 Machine Learning Model Definition

Creating a machine learning model typically involves the following steps (Aurélien Géron, 2019):

1. **Collect and prepare the data**

   Gather a dataset that is representative of the problem you want to solve, and then clean and preprocess the data so that it can be used to train a model.

2. **Choose a model**

   Select a model architecture that is appropriate for the problem and the data. Common choices include decision trees, random forests, and neural networks.

3. **Train the model**

   Use a subset of the data to train the model. This typically involves providing the model with input data and corresponding correct outputs and adjusting the model's parameters so that it can accurately predict the output for new input data.

4. **Evaluate the model**

   Use a separate subset of the data to evaluate the model's performance. This typically involves providing the model with input data and comparing its predictions to the true output.

5. **Fine-tune and improve the model**

   Based on the evaluation, fine-tune the model by adjusting its parameters and trying different model architectures until you are satisfied with its performance.

2.3.3 Feature Engineering

As the name implies, Feature Engineering is the process of creating new features or modifying existing features from raw data to improve machine learning models' performance. It involves transforming raw data into a set of meaningful and relevant features that can better represent the underlying patterns and relationships in the data, leading to improved model accuracy and better prediction results (Heaton, 2017). It is a crucial step in the machine learning pipeline typically found in step 1 of creating a model above, as the quality and relevance of the features can greatly impact the performance of the models. It involves using domain knowledge and understanding of the problem to extract useful information from raw data and convert it into a set of features that can be fed into a machine learning algorithm. This process can includes:

- **Feature selection:** Selecting a subset of the available features to use in the model, based on their importance or relevance.
- **Feature extraction:** Creating new features from existing features using mathematical operations or transformations.
- **Feature scaling:** Normalizing the values of the features so that they are on the same scale, can improve the performance of certain algorithms.
- **Feature encoding:** Converting categorical variables into numerical variables so they can be used by machine learning algorithms.

The goal is to provide the machine learning model with a better representation of the data, allowing it to make more accurate predictions. It requires a deep understanding of the problem and the data and often requires trial and error to determine the most effective set of features.

2.3.4 K-fold Cross-Validation

K-fold Cross-Validation is a technique that allows us to estimate the performance of a model on unseen data by partitioning the available data into K subsets or folds (James et al., 2013). The model is trained on K-1 folds and evaluated on the remaining fold. This process is repeated K times, with each fold used once for an evaluation. The performance of the model is then averaged over the K iterations to obtain a more reliable estimate of its performance.

The process of K-fold Cross-Validation can be summarized as follows:

1. Partition the data into K subsets or folds.
2. For each iteration, select one fold as the test set and use the remaining K-1 folds as the training set.
3. Train the model on the training set and evaluate it on the test set.

4. Repeat the process K times, with each fold used once as the test set.
5. Calculate the average performance of the model over the K iterations.

This validation provides several benefits over other model evaluation techniques, such as hold-out validation. One of the main benefits is that it allows using all available data for both training and evaluation. This is important because it maximizes the use of available data and reduces the risk of overfitting. K-fold Cross-Validation also provides a more reliable estimate of model performance by averaging the results over multiple iterations. This reduces the impact of random variations in the data and provides a more accurate estimate of model performance.

2.4 Related Work

This section will review related work in game analytics and dynamic adjustment systems, highlighting the significant contributions made in this field.

2.4.1 Game Analytics in Player Behavior

In recent years, there has been a growing interest in using game analytics to better understand player behavior. Several studies have been conducted to analyze the massive amounts of data generated by gaming platforms such as Steam. In this section, we will review three key papers that have contributed to the field of game analytics for player behavior.

Sifa et al. (2014) conducted a large-scale cross-game player behavior analysis on Steam, which aimed to understand how players engage with different games. The study analyzed a dataset of over 20 million players and identified different player types based on their behavior. The authors used unsupervised machine learning algorithms to cluster players and identified several distinct groups such as "completionists," "explorers," and "socializers." The study also revealed insights

into the playing habits of different player types, including how long they played games, their purchase behavior, and how they interacted with game communities.

Baumann et al. (2020) conducted a study that used unsupervised learning to analyze the behavior of hardcore gamers on Steam. The study aimed to identify patterns in the playing behavior of players who spend a significant amount of time on games. The authors analyzed a dataset of over 15 million users and identified different types of hardcore gamers based on their behavior. The study revealed several key insights, including that hardcore gamers tend to focus on specific genres of games and have a preference for playing solo. The study also highlighted the importance of understanding the behavior of hardcore gamers for game design and development.

Sifa et al. (2021) conducted a study that used the playtime principle to model player interest in games. The study analyzed a dataset of over 100,000 games on Steam and identified different types of games based on the amount of playtime. The authors used unsupervised learning algorithms to cluster games and identified several distinct groups such as "instant gratification" and "long-term engagement." The study also revealed insights into the playing habits of different player types, including how long they played games, their purchase behavior, and how they interacted with game communities.

Overall, these studies demonstrate the power of game analytics for understanding player behavior. By using large-scale datasets and advanced machine-learning algorithms, researchers can gain valuable insights into how players engage with games. These insights can inform game design and development and lead to more engaging and successful games.

2.4.2 Dynamic Difficulty Systems

As AI technology progresses, there is an increasing interest in direct game AI control for DDA approaches to provide more effective game difficulty adjustment (Demediuk et al., 2017, Ishihara et al., 2018, Moon and Seo, 2020, Zohaib and Nakanishi, 2018). The majority of these studies focus on modifying game aspects for game complexity by focusing on the player's expertise.

There are various heuristic methods that have been employed to evaluate a player's performance, such as analyzing the success rate (as seen in studies by Duque et al. 2020 and Sarkar and Cooper, 2019), game scores (examined by Hagelback and Johansson, 2009, Silva et al., 2017) and health points (HP) (investigated by Demediuk et al., 2017, Ishihara et al., 2018). For instance, in games that feature opponent agents such as fighting games, the difficulty level can be adjusted by modifying the skills of the game agents through techniques like tree search (as seen in studies by Demediuk et al., 2017, Ishihara et al., 2018) or deep learning (studied by Pfau et al., 2020) based on heuristic metrics such as HP-difference.

2.5 Research Gap

Most of the current research on DDAs has shown great advances using the power of Machine Learning and Deep Learning. However, they focus mainly on using the current game state variables of the player like game score or health points to be able to adjust the difficulty, which could show relatively static and predictable difficulty alterations. Every player has different difficult expectations. For instance, some players would like challenging games, whereas others like relaxed, winnable games. Using heuristic adjustments is too limited to satisfy every player's expectations.

Furthermore, there is a huge amount of data that could be used beforehand to create a more accurate prediction of how a specific player will perform based on their experience in similar games. For instance, Steam, a PC gaming platform for buying games, has one of the biggest databases of players with over 120 million active players and 62+ million active users (Chang, 2021). Fortunately, most of this data is available to any 3rd party developer via a REST API. It allows querying data such as current games owned, total time played on a specific game, and achievements. Using this data could lead to different datasets that solve all kinds of problems, including giving information on prior play experience that could be used to start players with a better current play experience This will be examined further in the next chapter.

Chapter 3 – Research

As mentioned in Chapter 1 of this thesis, our primary goal is to test the potential of our model to correctly assess the ability of players by using previous gaming information to create a pre-assessment module that will improve the initial state of DDAs.

In this chapter, we will define the general architecture, navigate the dataset generation and review the analysis and data processing to create the dataset of players. We have identified these as the necessary initial steps to train and test our machine-learning algorithms.

3.1 Architecture

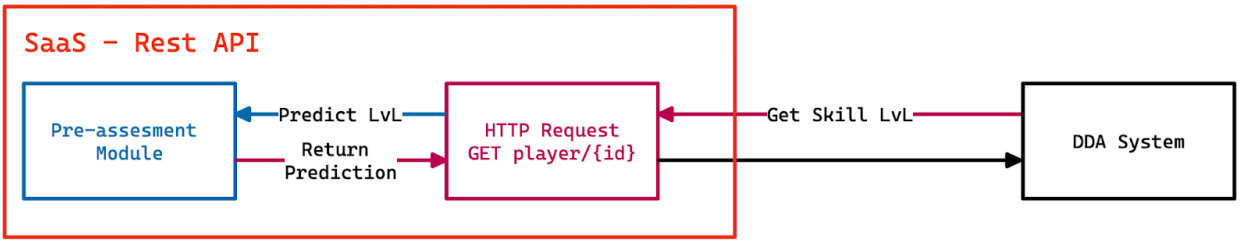

Figure 1. Pre-Assessment module integration

Our overall architecture is comprised of three important parts, as shown in Figure 1. First is a pre-assessment module that contains a machine learning model used to predict the player skill level based on data related to previous play. Second is a REST API that provides an agnostic bridge between the pre-assessment module and the DDA system. Finally, the DDA system is in charge of modifying the difficulty of the game, leveraging feedback from the pre-assessment module to start the player at a more optimal initial difficulty setting.

3.1.1 Pre-Assessment Module

Our primary goal is to demonstrate the potential of using previous gaming information to create a pre-assessment module that will communicate with any game or system. To achieve this we divided this task into four separate sections. First, is the dataset generation, which involves scraping different Steam users to obtain their previous gaming information and store it in an SQLite database for further analysis. Second, in data analysis and dataset creation, we analyzed the players, achievements, and games to be able to create different features that help our model's performance and accuracy. Third, we trained an unsupervised learning algorithm (K-means) to understand better the data with three different clusters that determine which difficulty level a player should be playing for a specific genre. Finally, we can generate the dataset with the new difficulty label and train our model on different algorithms to determine which performs better. Once the model is trained and tested, we save it and deploy a web service that can be used by any game or any type of application that has a connection to the internet.

3.1.2 SaaS - REST API Integration

The purpose of this SaaS application is to provide Dynamic Difficulty System and game developers with useful information about players at the start of their game, regardless of the game engine or programming language used.

Software as a Service is a cloud-based software delivery model in which the cloud provider develops and maintains cloud application software, provides automatic software updates, and makes software available to its customers via the internet on a pay-as-you-go basis. The public cloud provider manages all the hardware and traditional software, including middleware,

application software, and security. So SaaS customers can dramatically lower costs; deploy, scale, and upgrade business solutions more quickly than if they were maintaining on-premises systems and software; and predict the total cost of ownership with outstanding accuracy (Learn about SaaS, 2020).

This application can be deployed on any cloud provider as a RESTful API, which enables game developers to make HTTP requests to the server to obtain data from the machine learning model in JSON format. The application requires the player identifier (Player ID) and the platform on which their profile information lives (Steam, Epic, Xbox, Playstation, etc.), and returns a response with the player skill level.

3.1.3 Dynamic Difficulty System

The DDA system, which is integrated directly with the games and responsible for manipulating the difficulty, constitutes the final component of the overall architecture. By utilizing our pre-assessment module to seed its initial state, the DDA system can offer users a more optimal starting difficulty.

3.2 Genre Focus

For a manageable initial data set, we have focused on a particular genre of games available through the Steam storefront. Specifically, we manually curated a list of 25 games from the shooter genre, retrieved from the Steam website filtered by different tags (Action[1] FPS[2], FPS, Shooter[3]) related to shooter games. Afterwards, we extracted basic information such as game name and ID, stored in a JSON file. This file will serve as a reference point for the research as they gather all the necessary data to train the machine learning model.

Table 1. Games List

	Game	Steam Code
1	Battlefield 1	1238840
2	Battlefield 4	1238860
3	Battlefield 5	1238810
4	Battlefield 2042	1517290
5	Call of Duty Advanced Warfare Multiplayer	209650
6	Call of Duty Modern Warfare 3	115300
7	Call of Duty Modern Warfare 2	10190
8	Call of Duty WWII Multiplayer	476600
9	Call of Duty: Black Ops 3	311210
10	Call of Duty: Black Ops II	202970
11	Call of Duty Black Ops II Multiplayer	202990
12	Tom Clancy's Rainbow Six® Siege	359550
13	Call of Duty Ghosts Multiplayer	209160

1 https://store.steampowered.com/category/action_fps/

2 https://store.steampowered.com/tags/en/FPS/

3 https://store.steampowered.com/tags/en/Shooter

14	Call of Duty®: Infinite Warfare	292730
15	PUBG: Battlegrounds	578080
16	Destiny 2	1085660
17	Apex Legends	1172470
18	Halo: The Master Chief Collection	976730
19	Halo Infinite	124044
20	Titanfall 2	1237970
21	Metro Exodus	412020
22	Metro Last Light Redux	287390
23	Metro 2033 Redux	286690
24	Borderlands 3	397540
25	Counter-Strike: Global Offensive	730

3.3 Dataset Generation

We decided to generate our dataset based on the Steam achievement tracking website [4]and the Steam Application interface[5].

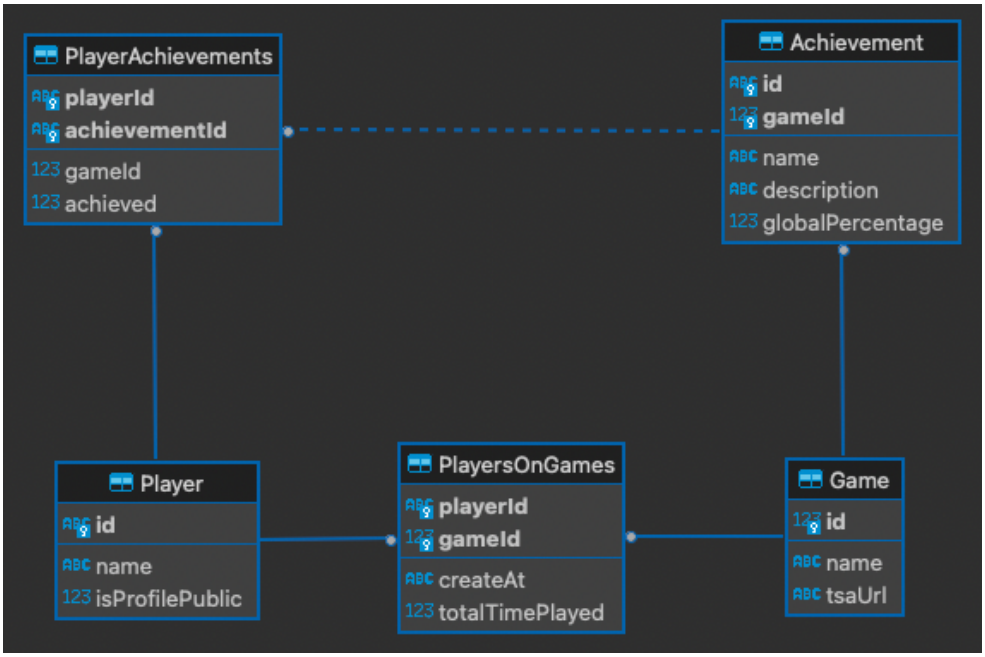

Figure 2. Entity Relation Diagram for Web Scraper

Initially, we defined an Entity Relation Diagram (see Figure 2.) that relates all primary entities of our model (Player, Game, and Achievement). We will elaborate on the details in the section that follows.

4 https://truesteamachievements.com/

5 https://partner.steamgames.com/doc/webapi

3.3.1 Database Entities

Our player entity table contains all the attributes of a player. The most important field is the Player ID, which allows us to relate the player with our other entities.

Table 2. Player Entity fields

Player Entity	
id	Unique Player identifier across the database
name	Player nickname
isProfilePublic	Flag to know if the player information is publicly available

Our game entity table contains all the attributes from a game. Similar to the Player Entity, we use the Game ID to relate different games to players and achievements.

Table 3. Games Entity Fields

Game Entity	
id	Unique Game identifier across the database
name	Game name
tsaUrl	Url to get extra information about the game

In games, an achievement is a goal or objective that a player can accomplish within the game. Achievements are often associated with specific actions or milestones that the player must reach, such as completing a level, defeating a boss, or collecting a certain number of items. Achievements can serve as a way for players to track their progress and demonstrate their skill or dedication to the game, as well as provide additional challenges beyond the core gameplay.

The achievement entity table contains all the related attributes for a specific achievement of a game provided by the Steam API.

Table 4. Achievement Entity Fields

Achievement Entity	
id	Unique Achievement identifier across the database
gameId	Unique Game identifier across the database
name	Achievement name
description	Short description of the achievement task
globalPercentage	Percentage of people in Steam that own the game and unlock the achievement.

The Player on Games table represents a Many-to-Many relationship between players and games. Specifically, it enables the identification of games owned by a particular player from Table 2. Additionally, it has the capability to store various statistics and data provided by the Steam API that pertain to a player's interaction with a game, such as TotalTimePlayed.

Table 5. PlayerOnGames - Relationship between Player and Games

PlayerOnGames	
playerId	Unique Player identifier across the database
gameId	Unique Game identifier across the database
createdAt	Time web this record was created
totalTimePlayed	Player total time played on a specific game

The Player Achievements table is generated by the Many to Many relationships between the Player and Achievements. Mainly it contains if the player has unlocked the achievement on a specific game.

Table 6. Player-Achievements - Relationship between Player and Achievements

PlayerAchievements	
playerId	Unique Player identifier across the database
AchievementId	Unique Achievement identifier across the database
gameId	Unique Game identifier across the database
achieved	Number between 0 and 1 to determine if the player unlocked the achievement.

3.3.1.1 Web Scraping and Data Fetching

Even though the Steam application programming interface provides all the information regarding Games and Achievements, when dealing with Players' information it becomes more complicated. Players' information is not public unless the player agrees to make their profile public. At the same time, we need the player ID to interact with the API, which is not available directly from the Steam website.

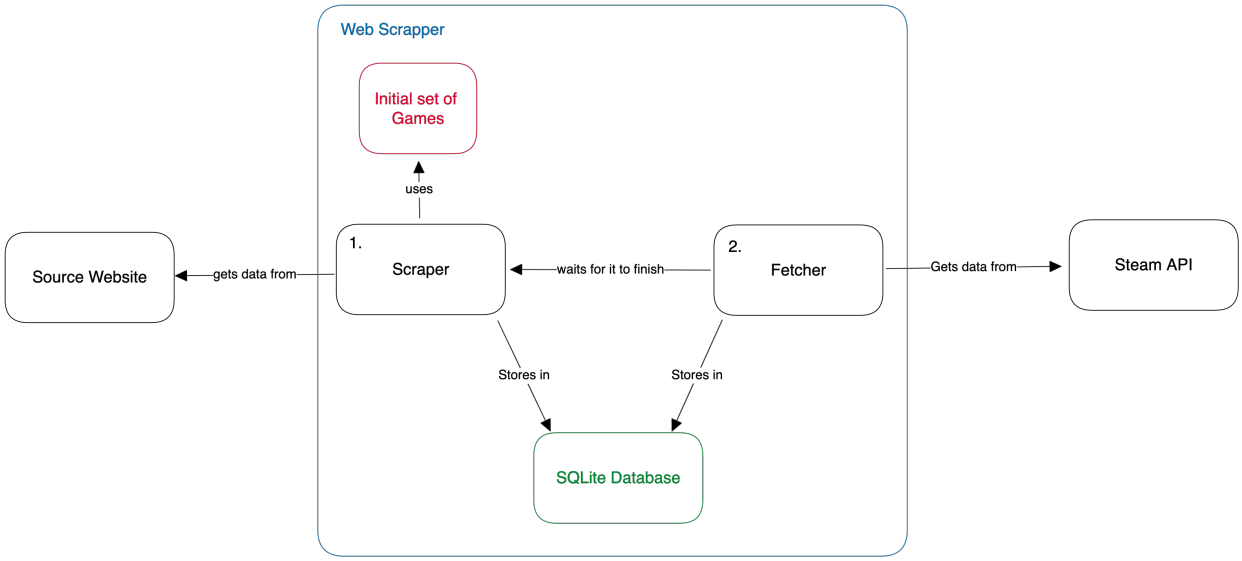

Figure 3.Web Scraper implementation diagram

To bypass these limitations, we created a Web Scraper (see Figure 3) that is in charge of retrieving mainly the required player information to interact with the Steam API (Player ID). The Scraper is subdivided into two important elements that work together synchronously to retrieve all the data from the source website:

- **Scraper:** Responsible for obtaining player information, including the player's identification and name, from the source website[6]. This is achieved by utilizing a web automation tool called Puppeteer[7], which interacts with the Chrome web browser, to gather all the data.

- **Fetcher:** Responsible for acquiring player, game, and achievement information from the Steam API and storing it in an SQLite database. It must wait for the Scraper to complete its task to ensure that it has all the necessary information to begin.

Having defined the responsibility of each element, we can define the following steps for retrieving data:

1. Utilize the initial list of 25 games as a starting point.
2. Retrieve information on all the achievements associated with each game.
3. Iterate through each game and scrape the achievements leaderboard to acquire approximately 100 player identifiers.
4. Using these player IDs, the scraper can then iterate through each game and each player to collect information on a specific player's achievements in a particular game from the platform application interface, in this case, the Steam API.

Once it performs all the steps, the Scraper stops, and all the data is stored in a local SQLite database. Even though the Scraper currently only takes the shooter genre into account, it only needs a small modification to get information about any game.

[6] https://truesteamachievements.com/
[7] https://pptr.dev/

Once it performs all the steps, the Scraper stops, and all the data is stored in a local SQLite database. Even though the Scraper currently only takes the shooter genre into account, it only needs a small modification to get information about any game.

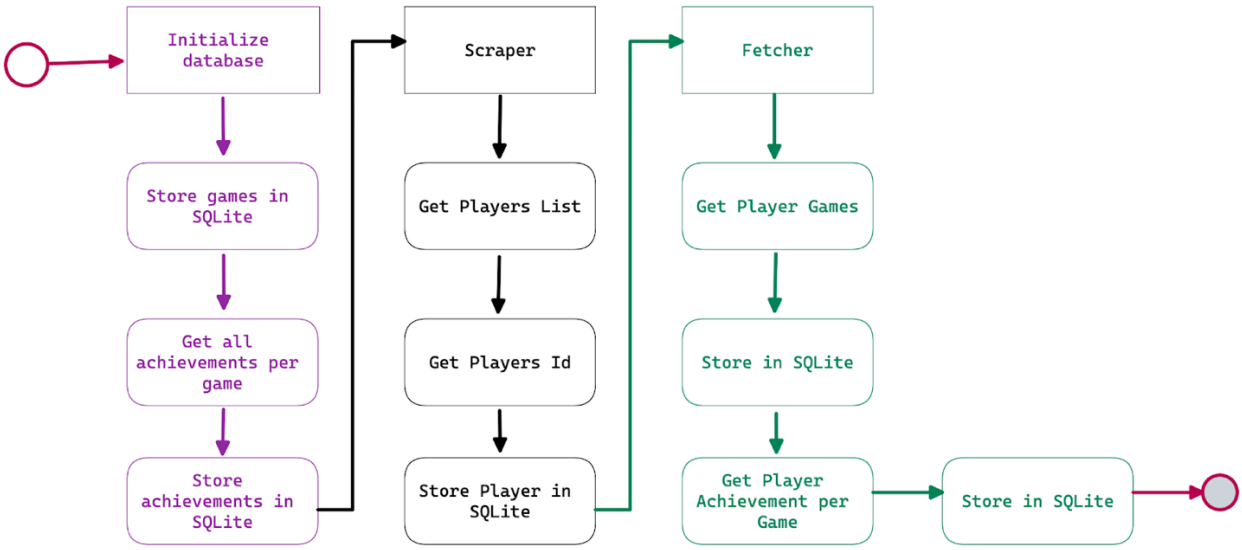

Figure 4. Web Scraping and Data fetching processes.

3.3.3 Data Metrics

Table 7. Data Metrics

Entity	Total
Games	25
Players	11,925
Achievements	2,193
Player-Achievements	~ 6.5M and only ~1.6M achieved
Player-Games	81,516

We were able to retrieve a substantial amount of data from the Steam API, including information on players, achievements, and games. However, we encountered a limitation in the Steam API, as it did not provide complete data for certain player and game combinations. Specifically, we found that only a subset of achievements was retrieved, with all the unlocked ones and some locked ones. This posed a challenge as we expected to retrieve approximately 650 million player achievements. Despite this limitation, we were able to extract a considerable amount of relevant data for our analysis. That said, the missing values would not affect the end result of the study since we retrieve all the unlocked achievements.[8]

[8] https://developer.valvesoftware.com/wiki/Steam_Web_API#GetPlayerAchievements_.28v0001.29

3.3.4 Data Cleaning

To perform an accurate analysis, it is essential to ensure that the dataset being used is reliable and relevant. To achieve this, we executed various queries directly on the database, filtering out any unnecessary data such as players with zero time played, or no achievements on a specific game. That way, we only retrieve the information required for the analysis and the model, storing each of the results in a separate CSV file for future manipulation. By doing so, we were able to generate a clean dataset that is tailored to the specific needs. This approach not only ensures the accuracy of our dataset but also saves time and computational resources by excluding irrelevant data from the analysis. The use of a clean dataset is critical in ensuring validity and reliability.

3.4 Feature Engineering

For this thesis, we use different feature engineering techniques to create a new feature that provides a better insight into the relationship between Players, Achievements, and Games. Once we analyzed the data, we were able to select five existing features and create six new ones.

3.4.1 Feature Selection

We can get multiple relevant features from information already provided by the Steam API. The following features provide some insight into how the player like to invest their time in gaming:

1. *Total Games Owned:* Amount of games owned by the player
2. *Total Time Played:* Amount of time spent by the player overall on the owned games
3. *Total Percent Completed*: How invested is the player in unlocking achievements
4. *Total Achievements*: Amount of achievement in all games
5. *Average Achieved*: Average unlock achievements in all games.

3.4.2 Feature Extraction

Initially, there is no quantitative value that we can use to classify achievements. That said, we will use a score function eq. (1) to give our achievements a value that the model can understand.

Using the achievements score as our base for feature extraction, we can create other features that represent the level of difficulty per achievement and subsequently per game.

1. *Total Achievement Score*: Total score over all owned games.
2. *Total Easy Achievements*: Amount of easy achievements unlocked owned games.
3. *Total Normal Achievements*: Amount of normal achievements unlocked owned games.
4. *Total Hard Achievements*: Amount of hard achievements unlocked owned games.
5. *Total Achieved*: Amount of achievements unlocked by the player in all games.
6. *Percentage Difficulty:* Addition of all achievement percent_difficulty of that specific player in all games.

The following section of this thesis will provide a detailed account of the process used to produce the achievement score, as well as all the associated features that were derived from it.

3.4.2.1 Achievement Score

The source of our player's data describes an interesting way to define the score function for different achievements.

$$Ratio = \sqrt{\frac{Players\ that\ Own\ the\ Game}{Players\ that\ unlocked\ the\ Achievement}} \quad (1)$$

The ratio is our base value for the achievement score eq. (1). It relates to the players that have the game and the player that has unlocked the achievement; that way we can separate the easily achieved ones from the more complex ones. Since we do not have access to all players in steam, Is important to know that this ratio will only use information from within our dataset. This means that the score can change over time when more people can unlock it or more players are added to the dataset (TrueGaming Network, 2015).

$$Achievement\ Score = 10 * Ratio \quad (2)$$

Unlike other platforms (Xbox, PlayStation), Steam does not have a base value or score for achievement (e.g., Trophies types in PlayStation); for that reason, we decided to use 10 as our base value for all achievements (TrueGaming Network, 2015).

Another new parameter we created was percent_difficulty, which determines the difficulty of individual achievement. This provides a quantitative value that we can use to classify achievement by difficulty.

$$Percent\ Difficulty\ (PD) = 10 \times \frac{Achievement\ Score}{\max\ (Game\ Achievement\ Score)} \quad (3)$$

eq. (3). Show how we calculate the difficulty for each achievement. To begin with, the base difficulty is determined by dividing the current score by the maximum score achieved in the same game. Afterward, we normalize the result to yield a value between 0 and 10. By doing this we set the achievement score in the context of their game instead of merely a numeric value.

To classify the achievement into one of three categories - easy, normal, or hard - we establish the following condition:

$$Easy = 0 < PD < 5$$

$$Normal = 5 < PD < 7$$

$$Hard = 7 < PD \leq 10$$

After conducting multiple tests in Chapter 4 using different ranges, we concluded that the ranges presented above provide a better separation of our clusters of players.

3.4.2.2 Achievement Pre-processing

In order to process and structure the data we used Python with several packages like Pandas[9], Numpy[10], Matplotlib[11], and Scikit-Learn[12].

As discussed in Section 3.3.4, the required data for our analysis was prepared in distinct CSV files. This enabled us to efficiently merge all the achievement queries, facilitating the subsequent calculation of both the score and difficulty level for each achievement.

```
SCORE_CONSTANT = 10
# Calculate score ratio
result['score_ratio'] = (np.sqrt(result['players_count']/result['total_achived']))
# Calculate achievement score
result['score'] = SCORE_CONSTANT * result['score_ratio']
# generate a dictionary with the max score of achievement per game
max_score_per_game = result.groupby('gameId')['score'].apply(np.max).to_dict()
# Calculate percentage_difficulty = 10 * (score / max(score))
result['percent_difficulty'] = result.apply(lambda x: (x['score'] /
np.max(max_score_per_game[x['gameId']]) * 10), axis=1)
# Create a int version to plot
result['int_difficulty'] = np.round(result.percent_difficulty)
```

Figure 5. Achievement score and difficulty calculations

[9] https://pandas.pydata.org/
[10] https://numpy.org/
[11] https://matplotlib.org/
[12] https://scikit-learn.org/

We ended with the following values per achievement:

Table 8. Pre-process achievement final Data Frame

Column	Description
Achievement ID	Achievement identifier
game ID	Game Identifier
Total Achieved	Total number of players who unlocked the achievement
Name	Game name
Player count	The total number of players that owned the game
Total Achievements per game	Total number of achievement in the game
Avgerage Global percentage.	Average percentage from the global Steam population

By using these values in conjunction with the player information we can generate our dataset of players.

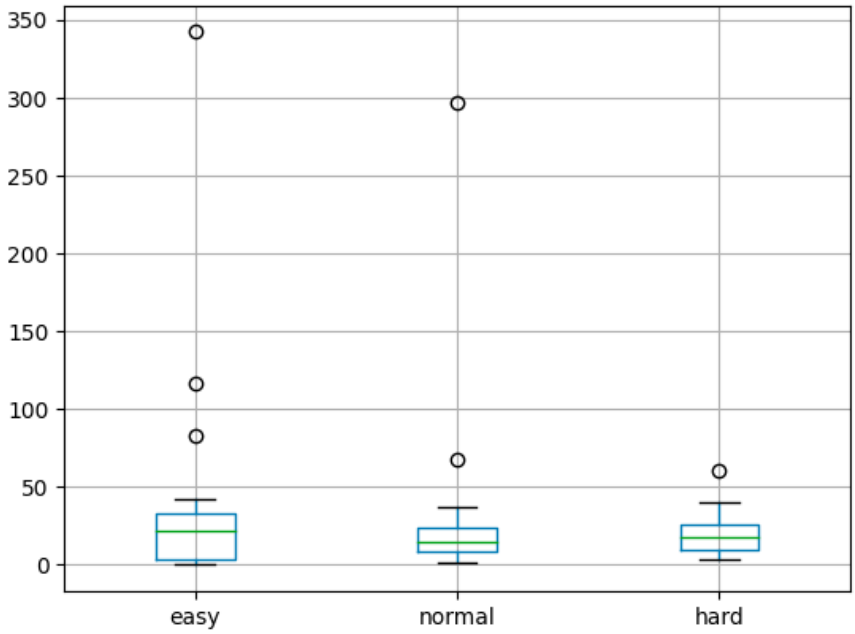

Figure 6. Achievement difficulty distribution

Based on the analysis of achievement distribution by difficulty level depicted in Figure 6, we can observe that a significant proportion of the achievements fall under the Easy category, followed by the Hard category, and lastly the Normal category. Notably, a few outliers are present, which can be attributed to the presence of a particular game (Halo: The Master Chief Collection) with 700 achievements. This finding sheds light on the distribution of achievements across different difficulty levels, providing insights into the gaming industry's trends and preferences.

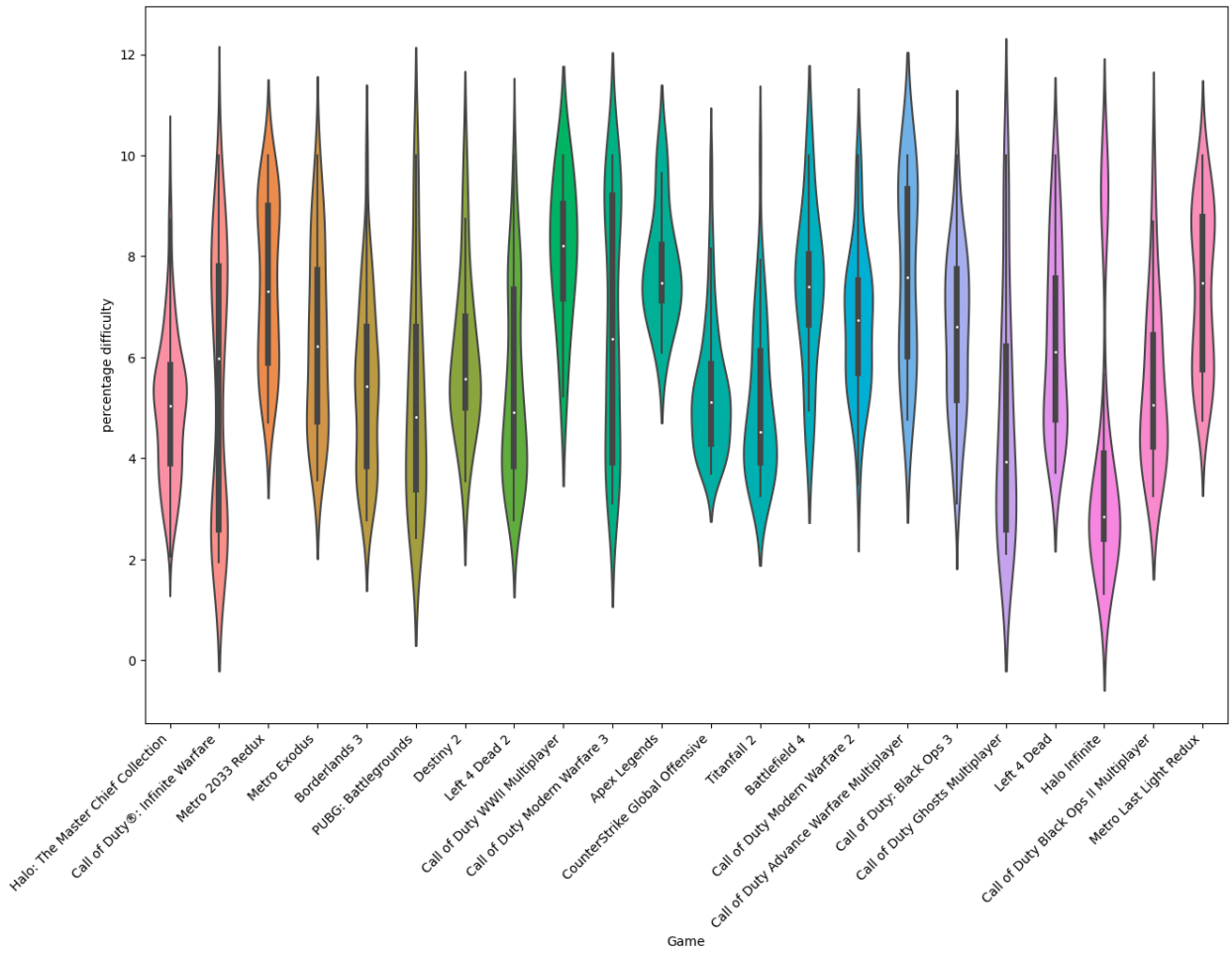

Figure 7.Violin Graph of achievements difficulty distribution across all games.

In Figure 7, we can see that the current data set contains a set of achievements in all the categories. Furthermore, the analysis shows that the average number of achievements per game tends to fall within the range of 4 and 8.

3.4.2.3 Players pre-processing

Once we finalize the achievements data from Section 3.4.2, we use the output with the independent achievements score and merge them with players' information. We performed different queries to our database to retrieve all the valuable information of the player as mentioned in Section 3.3.4.

Table 9. Preprocess player column output

Column	Description
Achievement_count	Total Achievements between all games
easy	Total Easy Achievements
normal	Total Normal Achievements
hard	Total Hard Achievements
total achieved	Total Amount of unlocked achievements
percent_difficulty	Addition of all achievement percent_difficulty of that specific player in all games.
avg score	Average achievements score
Total Achievement Score	Addition of all the unlocked achievement scores.
total time played	Total time across all games
game score	Amount of games times the avg_score
percentage_completed	total unlocked achievements divided by the total amount of achievements

Now that we have our dataset complete with all the features mentioned in Table 9, we can proceed into the next chapter where we train and test different Machine Learning algorithms to predict the player's skill level.

Chapter 4 - Experimentation

This chapter outlines the methodology and procedures employed in the experimental phase of the research. Our machine-learning model was defined, trained, and tested using the Python programming language and several open-source packages. The idea is to use a supervised model as the core for the pre-assessment module to predict the player skill level.

However, to be able to train a supervised model we need to have a label or target that we want to predict based on other features. In the absence of a labeled dataset, an unsupervised learning algorithm was utilized initially to cluster the data and analyze various player types. After successfully grouping and classifying the players, we proceeded to label the dataset and train multiple supervised learning algorithms looking to determine which of them performs better.

4.1 Unsupervised Learning - K-means for Player Classification

Once we have all features in place, we proceed to use the unsupervised machine learning method K-means to group our new dataset of players in different clusters and from them determine which difficulty is the most appropriate. For this, we will use Python's Sklearn as our machine-learning library.

```
from sklearn.compose import make_column_transformer
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder

# Create a column transfromar
ct = make_column_transformer(
    (MinMaxScaler(), players_data), # turn all values in these columns between 0 and 1
)

ct.fit(df)

X_train_norm = ct.transform(df)
```

Figure 8. Scaling players' data with SK-Learn

To use the K-means algorithm, we first had to scale the data to values between 0 and 1, since many machine learning algorithms use the Euclidean distance between two data points in their computations, and features with high values like *total_time_played* or *percent_difficulty,* can dominate the distance calculation.

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=3, random_state=RANDOM_SEED, n_init="auto")
label = kmeans.fit_predict(X_train_norm)
```

Figure 9. Training K-means

In Figure 9, we use our scaled data as input for the K-means algorithm with the following hyper-parameters:

Table 10. K-Means Hyper-Parameters

Parameter	Value	Description
n_clusters	3	Number of clusters
random_state	42	predefined constants to get similar results with each run
n_init	auto	Number of times the algorithm runs with different centroid seeds

We have defined the number of clusters in the k-means algorithm[13] as 3 to enable us to classify players' skill levels as either easy, normal, or hard. Additionally, we set the random state to 42 to ensure consistent results in each code execution. Furthermore, the default n_init parameter was employed, which runs the algorithm ten times using different centroid seeds and selects the best outcome.

[13] https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html

4.1.1 K-means Results

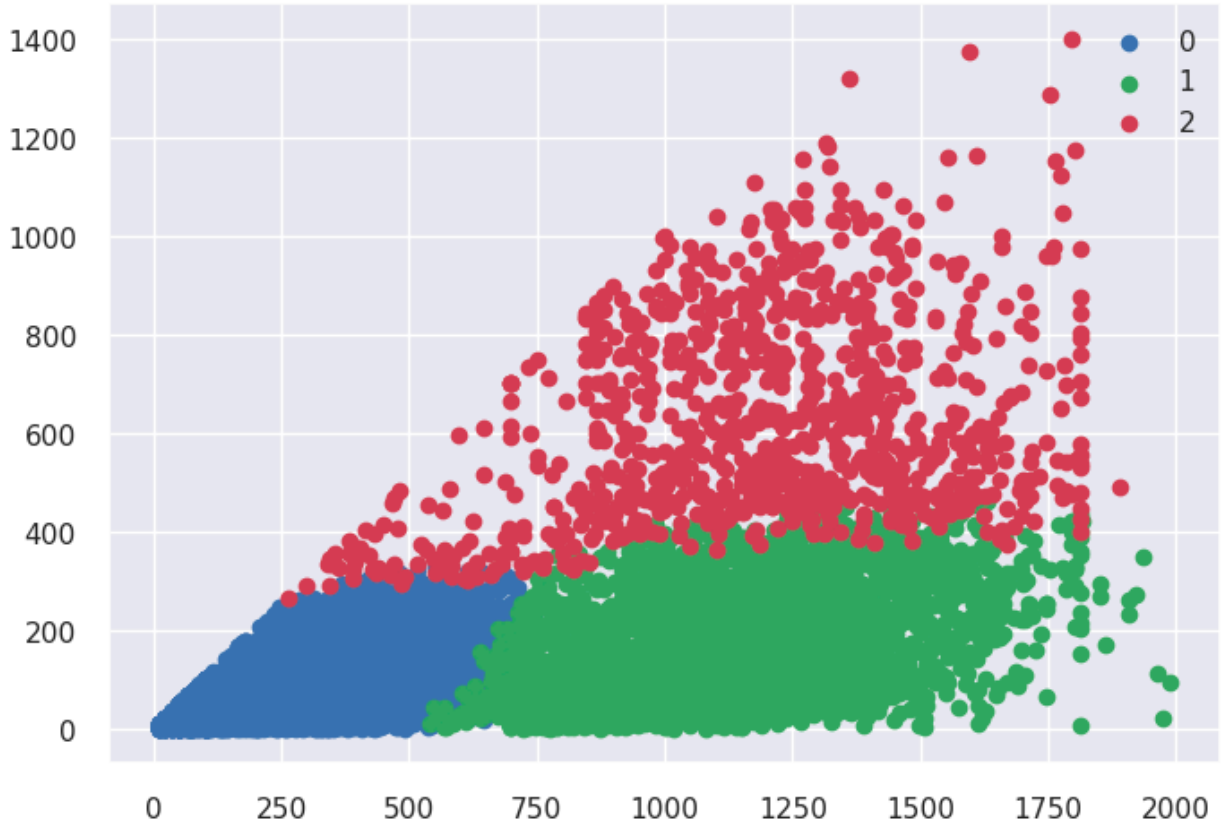

Figure 10. K-means Clusters

The results shown in Figure 10, show a clear separation of players by different clusters. To better understand what kind of players are in each cluster, we analyzed the following graphs.

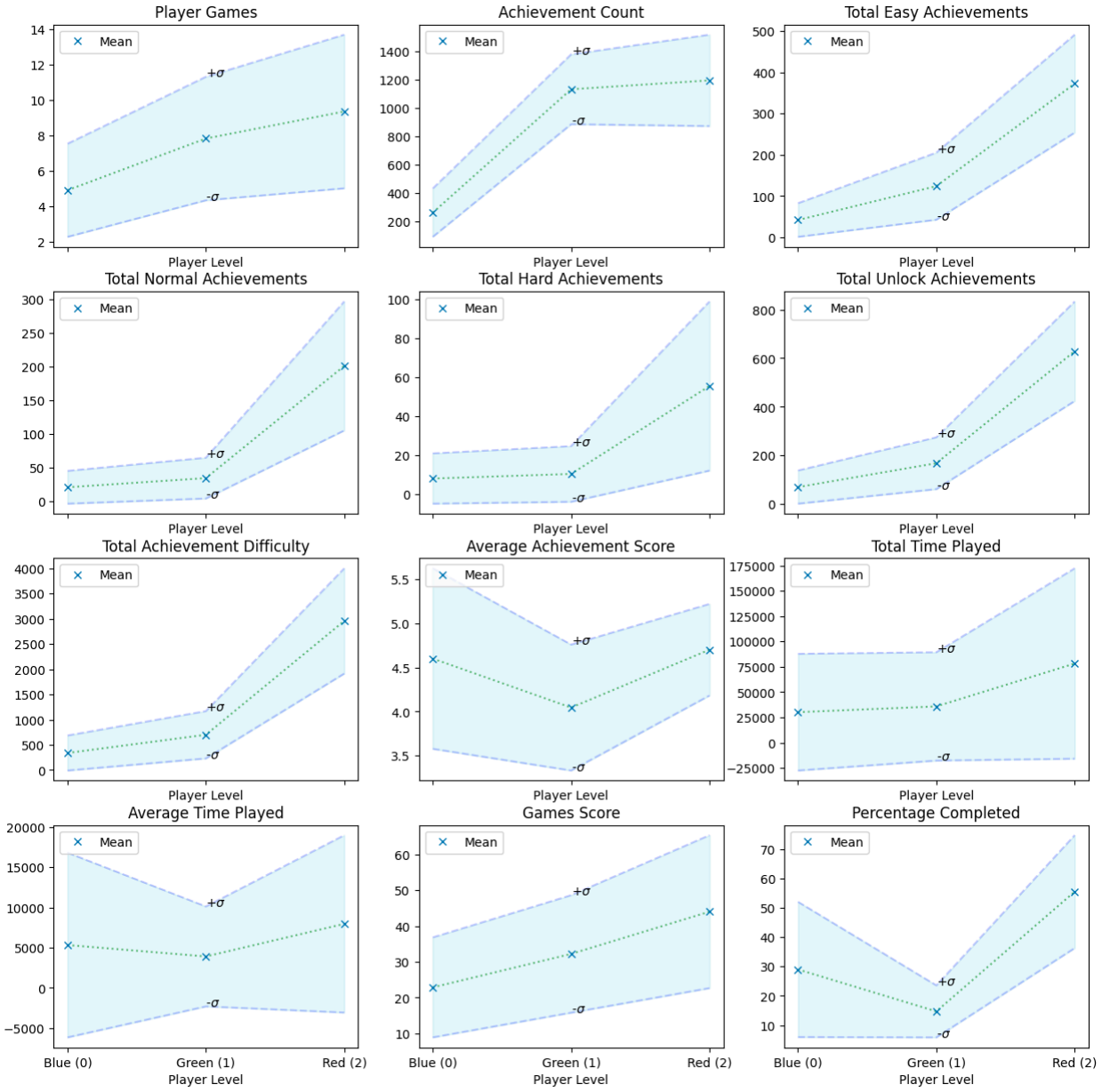

Figure 11. Player Features Mean-Standard Deviation Graph

Figure 11 shows all the player features with their mean values marked with an 'x' symbol and an area that describes the standard deviation. We can see that most of the features show an uptrend line that always puts the red players on the higher end, followed by the green players, and finally the blue players.

Upon analyzing Figure 10 and Figure 11, we determine that:

- The players categorized as Blue players had a low number of games and few achievements and therefore were considered easy players.
- The Green players were found to be more engaged with various games and had a moderate number of achievements, games played, and time invested in different games.
- The Red players were identified as the hard-core players, as they had the highest number of achievements per game and were driven to unlock as many achievements as possible.

After applying the unsupervised learning algorithm to analyze the dataset, we were able to identify patterns and groupings that allowed us to assign labels to the original dataset. This labeling was achieved by adding a new column to the dataset called *playerLevel*. This newly labeled dataset is used to train multiple supervised algorithms and test their performance.

```
df["playerLevel"] = pd.DataFrame(x_train).apply(
                    lambda x: kmeans.predict([x]).item()
                    , axis=1)
```

Figure 12. Adding playerLevel column to the original Data Frame

By using a variety of supervised learning techniques, we can compare and contrast their performance and select the most appropriate one for this particular study. This not only enhances the accuracy of our findings but also ensures that our analysis is rigorous and reliable.

4.2 Supervised Learning

```
from lazypredict.Supervised import LazyClassifier
from sklearn.model_selection import train_test_split

target = df_to_train.pop("playerLevel")

X_train, X_test, y_train, y_test = train_test_split(x_norm_to_train,
target, test_size=0.2, random_state=RANDOM_SEED)

clf = LazyClassifier(verbose=0, ignore_warnings=True, custom_metric=None)

models , predictions = clf.fit(X_train, X_test, y_train, y_test)
```

Figure 13. Initialize and train the LazyPredict package

To evaluate multiple supervised learning algorithms and conduct a comprehensive analysis of their overall performance, we employed the LazyPredict package. Initially, the target variable was defined as the newly created column “playerLevel”. Subsequently, we use Sklean’s “train_test_split” function to separate our dataset into train and test sets, with 80% and 20% of the data respectively. Following the acquisition of all necessary variables, the LazyClassifier module was initialized from the LazyPredict package, and both the training and testing data, together with their respective labels, were fitted.

Table 11. Results from LazyPredict

Model	Accuracy	Balanced Accuracy	F1 Score	Time Taken (s)
SGDClassifier	0.99	0.99	0.99	0.02
LogisticRegression	1.00	0.99	1.00	0.05
LGBMClassifier	0.99	0.99	0.99	0.41
XGBClassifier	0.99	0.99	0.99	0.98
RandomForestClassifier	0.99	0.99	0.99	0.63
ExtraTreesClassifier	0.99	0.98	0.99	0.22
BaggingClassifier	0.99	0.98	0.99	0.19
SVC	0.99	0.98	0.99	0.18
DecisionTreeClassifier	0.99	0.98	0.99	0.03
QuadraticDiscriminant Analysis	0.98	0.97	0.98	0.04
LinearSVC	0.99	0.97	0.99	0.13
Perceptron	0.99	0.97	0.99	0.02
LabelSpreading	0.98	0.96	0.98	1.35
LabelPropagation	0.98	0.96	0.98	0.96
GaussianNB	0.95	0.96	0.95	0.01

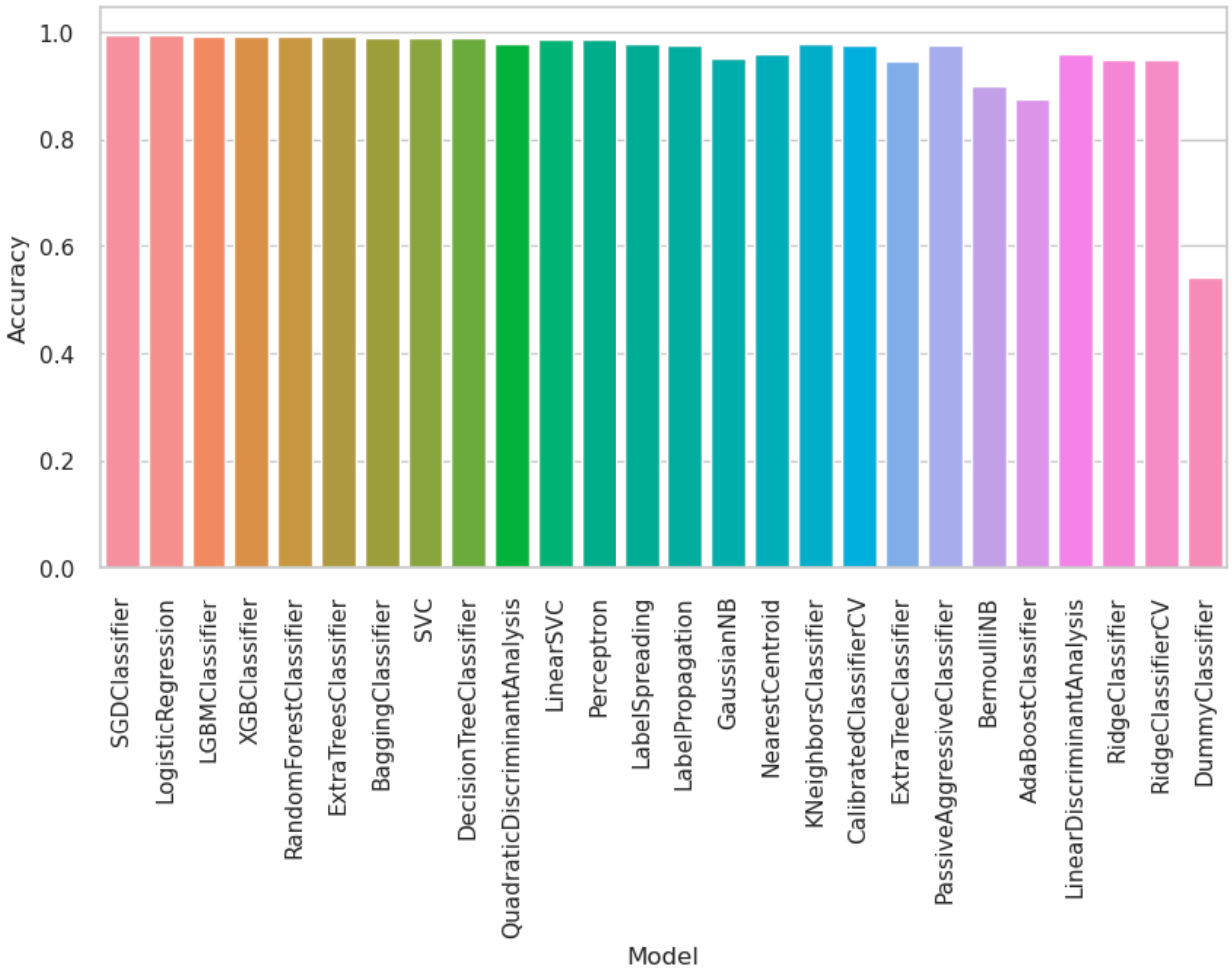

Figure 14. LazyPredict results chart.

Upon analyzing the results from the different algorithms, the SDGClassifier was the best for its high accuracy and short execution time. Prior to selecting an algorithm for our model, we seek to conduct a K-fold Cross-Validation to verify the model's accuracy regardless of which portion of the data is utilized as the test or training set.

Chapter 5 - Validation

Upon successfully creating a machine learning model based on our dataset, and obtaining positive results, we sought to conduct cross-validation to assess the model's ability to predict player difficulty levels regardless of the dataset used. To achieve this, we utilized the K-fold Cross-Validation technique, which involves dividing the data into K subsets, using each subset as a test set, and the remainder as a training set. This process is repeated K times to ensure that each subset is tested at least once, enabling a thorough evaluation of the model's performance.

5.1 K-fold Cross-Validation

Once more we use Python's Scikit library to perform the Cross Validation between our SGD Classifier and others already tested by the LazyPredict package (Logistic Regression, Linear SVM, Radial SVM, Decision Tree, Naive Bayes).

```
kfold = KFold(n_splits=10) # k=10, split the data into 10 equal parts
```

Figure 15. K-fold Cross-Validation class initialization

```
models=[
    SGDClassifier(random_state=RANDOM_SEED),
    LogisticRegression(max_iter=300, random_state=RANDOM_SEED),
    ...
]
for i in models:
    model = i
    cv_result = cross_val_score(model,x_norm_to_train, target, cv = kfold)
    cv_result=cv_result
    mean_accuracy.append(cv_result.mean())
    accuracy.append(cv_result)
```

Figure 16. 10-Fold Cross-validation with multiple models

First, we initialize our KFold Class with the value of n_splits or K, which in our case will be 10. Second, we loop through all of our models and use the cross_val_score function to perform the validation of our model. This function takes an initialized model, our features, the target label, and the cross validator, in our case, the already initialized KFold. Finally, we store the results of each model in two separate array variables, one with the mean value and another with the complete array of 10 scores.

Table 12. Average Model accuracy after using K-fold Cross-Validation

Model	Mean Accuracy Score of 10 subsets
SGDClassifier	0.97
Logistic Regression	0.98
Linear SVC	0.99
SVC	0.99
Decision Tree	0.99
Naive Bayes	0.94

Our result shows that the SGDClassifier is not, in fact, the best-performing algorithm when it comes to using different subsets of the data to test the algorithm.

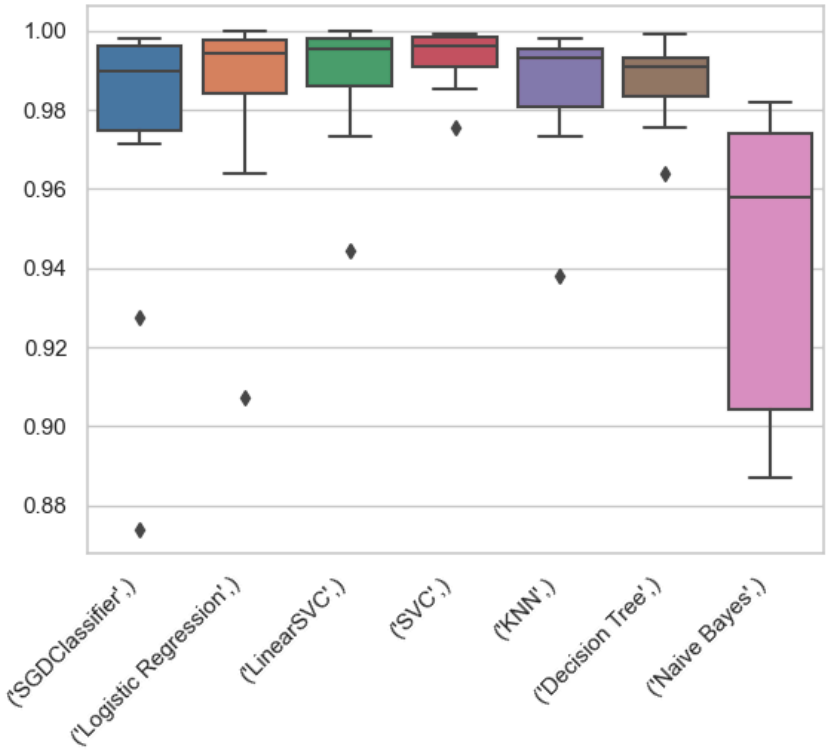

Figure 17. 10-Fold Cross Validation Result Graph

Further analysis of Figure 17 clearly shows that the SVC method has an overall better accuracy with our dataset.

5.2 Support Vector Classification (SVC)

To verify the Support Vector Classification (SVC) is the best fit for this use case, we did an isolated test directly with ScikitLearn package[14].

```
from sklearn import svm

clf_svc = svm.SVC(kernel='rbf', random_state=RANDOM_SEED)
clf_svc.fit(x_norm_to_train, target)
```

Figure 18. Support Vector Classification (SVC) Initialization

In Figure 18, we initialize our model with the random seed constant and then we proceed to train our SVC model.

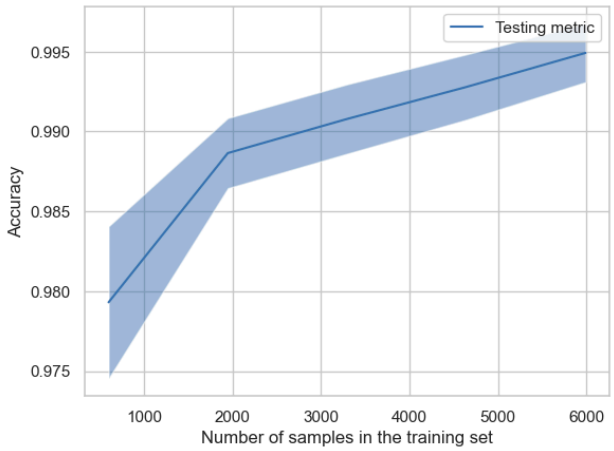

Figure 19. SVC Model Accuracy Chart

Upon examining the results in Figure 19, we can confirm that it gets similar results as Section 4.2.

[14] https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html

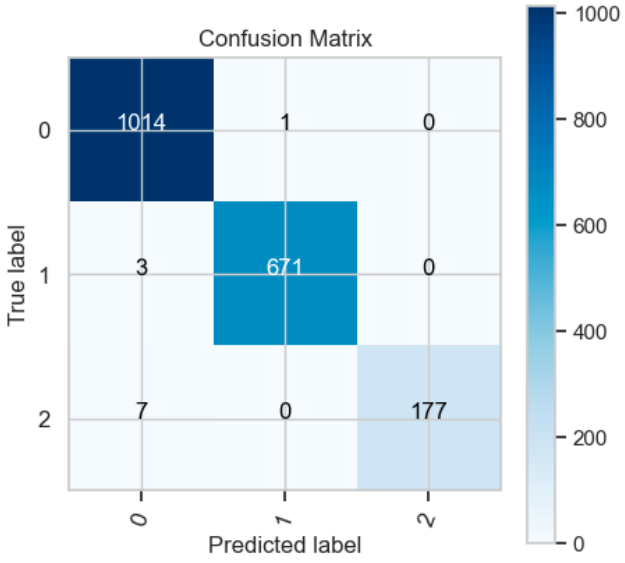

Figure 20. SVC Confusion Matrix

To further understand why the SVC classifier is having trouble classifying some players, we used a confusion matrix (see Figure 20). The results indicate that while the algorithm had no difficulty classifying easy players, it exhibited a tendency for missing predictions in the case of normal and hard players, usually as easy players. One plausible explanation for this phenomenon could be attributed to the insufficient representation of players at the normal and hard levels within the dataset.

5.3 Summary

Certainly, we can notice the potential and remarkable accuracy of using the player's past gaming data as a predictor of future performance in other related games. This predictive power allows for more appropriate initialization of game difficulty, tailored to each player's individual skill level. By using this pre-assessment method, players can be placed at a level of difficulty that both challenges and engages them, without being too easy or frustratingly difficult. Ultimately, this can lead to a more enjoyable and satisfying gaming experience for players.

Chapter 6 - Conclusion

6.1 Summary

The thesis aimed to explore a way to classify players' skill levels based on their previous gaming information. To achieve this, we generated a comprehensive dataset of players by extracting data from the Steam platform. Subsequently, the unsupervised method K-Means was used to create clusters of players based on unlocked achievements and games owned. The resulting clusters were used to label the original dataset and train multiple machine learning algorithms to later use the best one to accurately classify players into skill levels. Afterward, we choose the SGDClasssifier as our primary model and train it with our dataset. Finally, we used K-fold Cross-Validation to confirm that our algorithm can perform very well with any part of the dataset.

The findings of this study support the notion that leveraging previous gaming data can facilitate the creation of a pre-assessment module that can be integrated with a Dynamic Difficulty Adjustment (DDA) system or game. Such a module should improve the initial state of the game and enhance the player's overall experience.

6.2 Contribution

The present study has established a foundation for future research endeavors that aim to investigate the use of achievement-based metrics as a novel approach for characterizing player behavior and engagement in video games. By leveraging the extensive data archives maintained by platforms such as Steam, our method offers valuable insights into the abilities of individual players.

Through the generation of a new dataset of players, we conducted a comprehensive analysis of the data and were able to develop a classification model that demonstrates high theoretical accuracy

in predicting the appropriate level for each player. Further research efforts in this area are expected to yield significant practical advancements, thereby enabling DDA systems to achieve a more comfortable and tailored experience for players and to allow them to do so significantly earlier in a game's lifecycle.

6.3 Limitation

Although we were able to create our model with high accuracy, there are still some limitations to take into account for this first iteration of the model:

- In comparison with other studies (Baumann et al., 2018; Sifa et al., 2014, 2021), the dataset used for this research is relatively small. Since there's no application programming interface to get a list of players we had to scrape the players from a third-party website that didn't provide a substantial amount of players.
- The dataset only considers players with public profiles from the Steam platform, potentially introducing a small bias towards PC players.
- Since we use a predefined list of games of the shooter genre, the model would not be able to perform an accurate prediction if the player did not have one or more of these games.

As we continue our research, it is essential to recognize these limitations to ensure that our findings are accurate and reliable, and improved in time.

6.4 Future Directions

In this section, we discuss some possible future directions for our research:

A. Validate real players with a full system.

In order to validate the effectiveness of the pre-assessment module, it is crucial to create a game that integrates this module and provides an initial state to a Dynamic Difficulty Adjustment (DDA) system. By doing so, it becomes possible to test the module with real players and evaluate its impact on the gaming experience.

B. REST API for Agnostic integration

Incorporating Machine Learning models into a game can be tricky. We believe that the fastest way to integrate with any game or system is through a REST API. Using this architecture, we can serve our model to any game developer agnostic to the programming language or game engine.

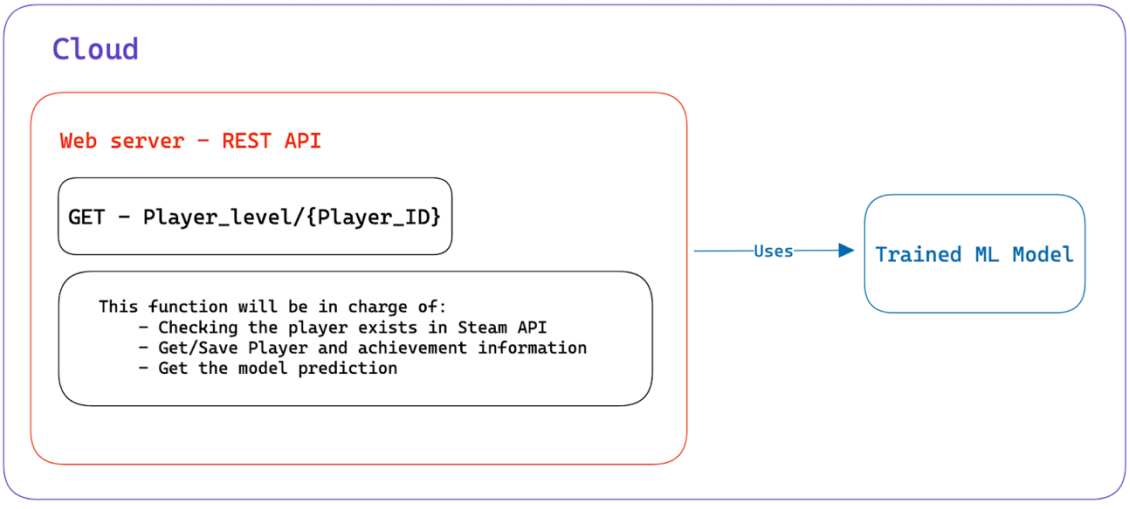

Figure 21. REST API Implementation Diagram

This endpoint should allow authenticated users to request a player skill level only by providing the player ID as a parameter. If the player is already stored, we only need to return the result from our database; otherwise, we need to retrieve the previous player information, generate an object with the player-specific parameters, and classify the player through our trained model.

C. Cross-Platform Integrategration

As a starting point, we used the Steam API to retrieve all the player's base data and achievements information but it is not the only platform with this kind of information. Xbox and Playstation represent almost 50% of the console/PC industry.

Even though Steam provides a great application programming interface to retrieve data from their database, other platforms such as Xbox and PlayStation have 120 million and 102 million monthly active users respectively (Sony, 2022; B. Iversen et al., personal communication, January 24, 2023). Integrating and analyzing the data from these platforms can provide better insight when dealing with console players and possibly an overall better accuracy on our model.

This integration would require either a slight modification of the database structure to determine the source of information or adapting the data from the other platforms to the current database structure.

D. Multi-Model Approach

The current implementation seeks to understand the player’s ability exclusively based on the achievements and their correlation with other features presented in this thesis. A multi-model approach could be used to understand all the entities in this study separately (games, players, and

achievements). Using the output from an Achievement and Game model as input into a Player model could lead to a tremendous increase in performance and accuracy. (Brownlee, 2021)

E. Tag-based Correlation for Games

Our research delimited the scope to only work with games from the shooter genre. As a future implementation, we could use user-defined tags from Steam and other platforms to correlate games.

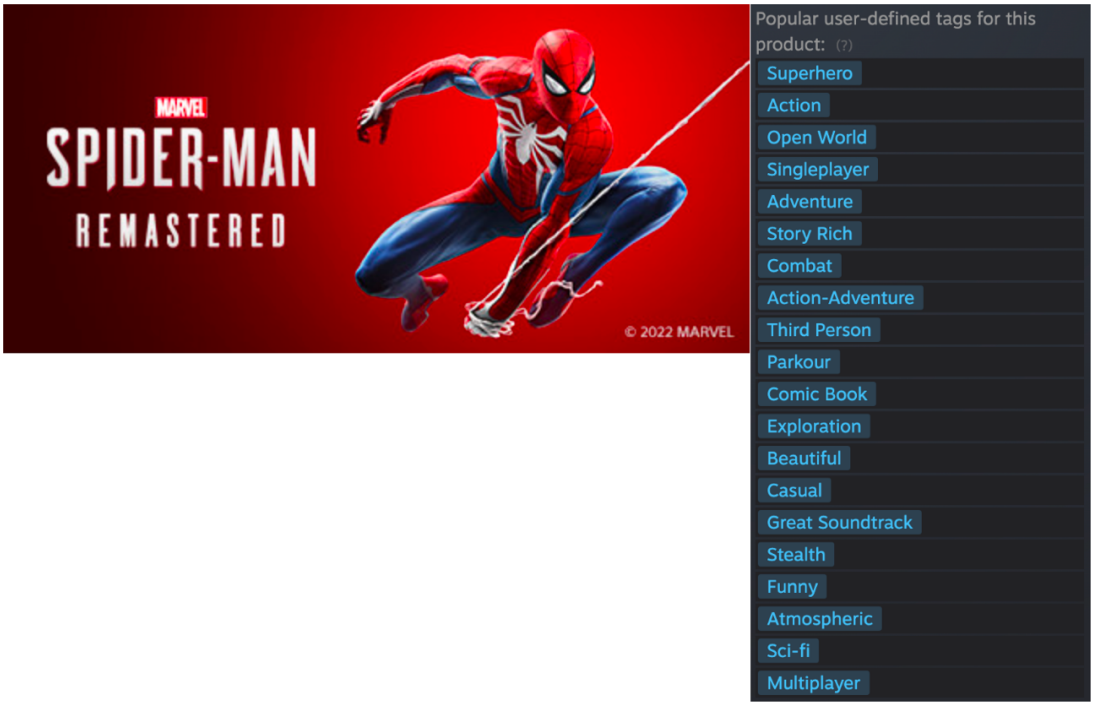

Figure 22. User-defined tags for Spider-man remastered from Steam.

For example, Spider-Man remastered[15] has 20 different tags defined by the platform users. Using the tags could provide an easier way to correlate games and even achievements.

[15] https://store.steampowered.com/app/1817070/Marvels_SpiderMan_Remastered/

This can be done by collecting data on the tags that users associate with each game and using that information to find similarities and relationships between the games.

Using clustering techniques, we can group similar items based on their similarities. In the case of games, the tags can be used as features, and the similarity between two games can be calculated based on the overlap of the tags they have been associated with.

Another approach is to use collaborative filtering, which finds correlations between the games that users have liked or played. In this case, the tags can be used as additional information to improve the accuracy of the recommendations.

F. Natural Language Model for Achievement Classification

Natural Language Model (NLP) is a field of artificial intelligence that focuses on the interaction between computers and human language and can be used to extract information and insights from large amounts of text data (Jain et al., 2018).

In the context of video game achievements, NLP can be used to analyze the descriptions and criteria for the achievements, understand the relationships between different achievements, and identify patterns and trends in player behavior. This information can be used to improve the design of the achievements and scoring system, provide recommendations to players based on their performance, or analyze player engagement and satisfaction.

For example, NLP techniques such as sentiment analysis could be used to determine the perceived difficulty or enjoyment level of specific achievements, while text classification could be used to categorize achievements based on their type (e.g. combat, puzzle, exploration).

Overall, NLP has the potential to greatly enhance our understanding of video game achievements and improve the gaming experience for players.

G. Time to Unlock

A feature that would be interesting to study is the time to unlock each achievement. Unfortunately, we would probably be working with an estimate instead of an exact value since we cannot tell when the player started to work on unlocking the achievement. That said, it could add some value to the overall achievement score and subsequently, improve the performance of the model.

References

Aurélien Géron. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. "O'Reilly Media, Inc."

Baumann, F., Emmert, D., Baumgartl, H., & Buettner, R. (2018). Hardcore Gamer Profiling: Results from an unsupervised learning approach to playing behavior on the Steam platform. *Procedia Computer Science*, *126*, 1289–1297. https://doi.org/10.1016/j.procs.2018.08.078

Brownlee, J. (2021, May 12). *A Gentle Introduction to Multiple-Model Machine Learning - MachineLearningMastery.com*. Machine Learning Mastery. Retrieved February 3, 2023, from https://machinelearningmastery.com/multiple-model-machine-learning/

Chang, J. (2023, January 9). *96 Steam Statistics You Must Know: 2022 Market Share Analysis & Data - Financesonline.com*. FinancesOnline.com. Retrieved January 15, 2023, from https://financesonline.com/steam-statistics/

Chang, J. (2021, March 30). *96 Steam Statistics You Must Know: 2021 Market Share Analysis & Data*. Financesonline.com. https://financesonline.com/steam-statistics/

Constant, T., & Levieux, G. (2019). Dynamic Difficulty Adjustment Impact on Players' Confidence. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems - CHI '19*. https://doi.org/10.1145/3290605.3300693

Demediuk, S., Tamassia, M., Raffe, W. L., Zambetta, F., Li, X., & Mueller, F. (2017). Monte Carlo tree search based algorithms for dynamic difficulty adjustment. *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. https://doi.org/10.1109/cig.2017.8080415

Drachen, A., Seif El-Nasr, M., & Canossa, A. (2013). Game Analytics – The Basics. *Game Analytics*, 13–40. https://doi.org/10.1007/978-1-4471-4769-5_2

Gilbert, N. (2020). *Number of Gamers Worldwide 2020: Demographics, Statistics, and Predictions - Financesonline.com*. Financesonline.com. https://financesonline.com/number-of-gamers-worldwide/

González-Duque, M., Palm, R. B., Ha, D., & Risi, S. (2020, August 1). *Finding Game Levels with the Right Difficulty in a Few Trials through Intelligent Trial-and-Error*. IEEE Xplore. https://doi.org/10.1109/CoG47356.2020.9231548

Hagelback, J., & Johansson, S. J. (2009). Measuring player experience on runtime dynamic difficulty scaling in an RTS game. *2009 IEEE Symposium on Computational Intelligence and Games*. https://doi.org/10.1109/cig.2009.5286494

Heaton, J. (2017, January 26). *[1701.07852] An Empirical Analysis of Feature Engineering for Predictive Modeling*. arXiv. Retrieved February 3, 2023, from https://arxiv.org/abs/1701.07852

IBM Cloud Education. (2020, July 15). *What is Machine Learning?* IBM. https://www.ibm.com/cloud/learn/machine-learning

Inal, Y., & Wake, J. (2022). An old game, new experience: exploring the effect of players' personal gameplay history on game experience. *Universal Access in the Information Society*. https://doi.org/10.1007/s10209-022-00872-0

Ishihara, M., Ito, S., Ishii, R., Harada, T., & Thawonmas, R. (2018, August 1). *Monte-Carlo Tree Search for Implementation of Dynamic Difficulty Adjustment Fighting Game AIs Having Believable Behaviors*. IEEE Xplore. https://doi.org/10.1109/CIG.2018.8490376

Iversen, B., Nadella, S., & Hood, A. (2023, January 24). *Microsoft FY23 Q2 Earnings Conference Call Transcript* [Conference Call]

Jain, A., Kulkarni, G., & Shah, V. (2018). Natural Language Processing. *International Journal of Computer Sciences and Engineering*, *6*(1), 161–167. https://doi.org/10.26438/ijcse/v6i1.161167

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An introduction to statistical learning : with applications in R*. Springer.

Learn about SaaS. (2020). Oracle.com. https://www.oracle.com/applications/what-is-saas/

Moon, J., Choi, Y., Park, T., Choi, J., Hong, J.-H., & Kim, K.-J. (2022). Diversifying dynamic difficulty adjustment agent by integrating player state models into Monte-Carlo tree search. *Expert Systems with Applications*, *205*, 117677. https://doi.org/10.1016/j.eswa.2022.117677

Pfau, J., Smeddinck, J. D., & Malaka, R. (2020). Enemy Within: Long-term Motivation Effects of Deep Player Behavior Models for Dynamic Difficulty Adjustment. *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3313831.3376423

Sarkar, A., & Cooper, S. (2019). Transforming Game Difficulty Curves using Function Composition. *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. https://doi.org/10.1145/3290605.3300781

Sifa, R., Bauckhage, C., & Drachen, A. (2014, August 1). *The Playtime Principle: Large-scale cross-games interest modeling*. IEEE Xplore. https://doi.org/10.1109/CIG.2014.6932906

Sifa, R., Drachen, A., & Bauckhage, C. (2021). Large-Scale Cross-Game Player Behavior Analysis on Steam. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, *11*(1), 198–204. https://doi.org/10.1609/aiide.v11i1.12804

Silva, M. P., Silva, V. do N., & Chaimowicz, L. (2017). Dynamic difficulty adjustment on MOBA games. *Entertainment Computing*, *18*, 103–123. https://doi.org/10.1016/j.entcom.2016.10.002

Sony. (2022). *Supplemental Information for the Consolidated Financial Results for the Second Quarter Ended September 30, 2022* (p. 9). Sony Group Corporation.

Togelius, J., Shaker, N., & Nelson, M. J. (2016). Introduction. *Procedural Content Generation in Games*, 1–15. https://doi.org/10.1007/978-3-319-42716-4_1

TrueGaming Network. (2015, December 24). *An Introduction To The TrueSteam Scoring System*. TrueSteamAchievements. Retrieved February 3, 2023, from https://truesteamachievements.com/n227/an-introduction-to-the-truesteam-scoring-system

Wijman, T. (2022, May 5). *Games Market Revenues Will Pass $200 Billion for the First Time in 2022 as the U.S. Overtakes China*. Newzoo. https://newzoo.com/insights/articles/games-market-revenues-will-pass-200-billion-for-the-first-time-in-2022-as-the-u-s-overtakes-china

Xue, Su, et al. "Dynamic Difficulty Adjustment for Maximized Engagement in Digital Games." *Proceedings of the 26th International Conference on World Wide Web Companion - WWW '17 Companion*, 2017, 10.1145/3041021.3054170.

Zagreb, F. (2018). *Machine Learning Based Models for Prediction of Player Behavior and Purchase Decisions in Digital Games*. https://muexlab.fer.hr/images/50033451/Filip%20Jurcic%20diplomski%202018.pdf

Zhou, C., Yu, X., Sun, J., & Yan, X. (2006, December 1). *Affective Computation Based NPC Behaviors Modeling*. IEEE Xplore. https://doi.org/10.1109/WI-IATW.2006.29

Zohaib, M. (2018). Dynamic Difficulty Adjustment (DDA) in Computer Games: A Review. *Advances in Human-Computer Interaction*, *2018*, 1–12. https://doi.org/10.1155/2018/5681652

Curriculum Vitae

Name:	Rafael David Segistan Canizales
Post-secondary Education and Degrees:	Technological University of Panama Panama, Panama 2014-2019 B.A.
	The University of Western Ontario London, Ontario, Canada 2021-2023 M.A.
Honours and Awards:	U.S. Panama Innovation and Competitiveness 2017
	UTP International – SMILE 2018-2019
	IFARHU-SENACYT – Canada 2020 Scholarship 2021-2023
Related Work Experience	Software Engineer Rootstack S.A 2019-2021