Pontifícia Universidade Católica
do Rio de Janeiro

Luís Fernando Teixeira Bicalho

A Generic Plugin for Player Classification in Games

Dissertação de Mestrado

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática.

Advisor: Prof. Bruno Feijó

Rio de Janeiro
July 2022

Pontifícia Universidade Católica
do Rio de Janeiro

Luís Fernando Teixeira Bicalho

A Generic Plugin for Player Classification in Games

Dissertation presented to the Programa de Pós–graduação em Informática of PUC-Rio in partial fulfillment of the requirements for the degree of Mestre em Informática. Approved by the Examination Committee:

Prof. Bruno Feijó
Advisor
Departamento de Informática – PUC-Rio

Prof. Augusto Cesar Espíndola Baffa
Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio

Prof. Alberto Barbosa Raposo
Pontifícia Universidade Católica do Rio de Janeiro – PUC-Rio

Rio de Janeiro, July 8th, 2022

Luís Fernando Teixeira Bicalho

Graduated in Computer Engineering at Pontifícia Universidade Católica do Rio de Janeiro, in 2019. Developed eletronic games as a member of one of ICAD Games/VisionLab student groups, winning the price for 2^{nd} best game of the fair in SBGames 2017. Is the author of two papers, one published in 2019 and other in 2020.

Bibliographic data

Bicalho, Luís Fernando Teixeira

A Generic Plugin for Player Classification in Games / Luís Fernando Teixeira Bicalho; orientador: Bruno Feijó. – 2022.

49 f: il. color. ; 30 cm

Dissertação (mestrado) - Pontifícia Universidade Católica do Rio de Janeiro, Departamento de Informática, 2022.

Inclui bibliografia

1. Classificação de Jogadores. 2. Modelos de Comportamento de Jogadores. 3. Telemetria. 4. Game Analytics. 5. Aprendizado de Máquina. I. Feijó, Bruno. II. Pontifícia Universidade Católica do Rio de Janeiro. Departamento de Informática. III. Título.

CDD: 004

Acknowledgments

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Abstract

Bicalho, Luís; Feijó, Bruno (Advisor). **A Generic Plugin for Player Classification in Games**. Rio de Janeiro, 2022. 49p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Game Analytics is an area that involves the processing of video game data, in order to make a better game experience for the user. It also helps to check the patterns in players behaviour, making it easier to identify the target audience. Gathering player data helps game developers identify problems earlier and know why players left the game or kept playing. These players' behavior usually follows a pattern, making them fit in different player profiles. Game analytics experts create and use models of player types, usually variants of Bartle's model, to help identify player profiles. These experts use clustering algorithms to separate players into different and identifiable groups, labeling each group with the profile type defined by the proposed model. The main goal of this project is to create a generic Unity plugin to help identify Player Profiles in games. This plugin uses a Python API, which deals with the game data stored in a MongoDB database, to cluster and label each match or level of the chosen game while the game is running. In this plugin, game developers can configure the number of player types they want to identify, the player labels, and even the algorithms they wish to use. This online clustering approach is not usual in game development. As far as we are aware, there is no software component in the game analytics literature with the same direction and features.

Keywords

Player Classification; Player Behavior Models; Telemetry; Game Analytics; Machine Learning.

Resumo

Bicalho, Luís; Feijó, Bruno. **Um Plugin Genérico para Classificação de Jogador em Jogos**. Rio de Janeiro, 2022. 49p. Dissertação de Mestrado – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

Game Analytics é uma área que envolve o processamento de dados de videogames com a finalidade de proporcionar uma melhor experiência de jogo para o usuário. Também ajuda a verificar os padrões de comportamento dos jogadores, facilitando a identificação do público-alvo. A coleta de dados dos jogadores ajuda os desenvolvedores de jogos a identificar problemas mais cedo e saber por que os jogadores deixaram o jogo ou continuaram jogando. O comportamento desses jogadores geralmente segue um padrão, fazendo com que se encaixem em diferentes perfis de jogadores. Especialistas em análise de jogos criam e usam modelos de tipos de jogadores, geralmente variantes do modelo de Bartle, para ajudar a identificar perfis de jogadores. Esses especialistas usam algoritmos de agrupamento para separar os jogadores em grupos diferentes e identificáveis, rotulando cada grupo com o tipo de perfil definido pelo modelo proposto. O objetivo principal deste projeto é criar um plugin Unity genérico para ajudar a identificar perfis de jogadores em jogos. Este plugin usa uma API Python, que lida com os dados do jogo armazenados em um banco de dados MongoDB, para agrupar e rotular cada partida ou nível do jogo escolhido enquanto o jogo está em execução. Neste plugin, os desenvolvedores de jogos podem configurar o número de tipos de jogadores que desejam identificar, os rótulos dos jogadores e até os algoritmos que desejam usar. Essa abordagem de agrupamento online não é usual no desenvolvimento de jogos. Até onde sabemos, não há nenhum componente de software na literatura de análise de jogos com a mesma direção e recursos.

Palavras-chave

Classificação de Jogadores; Modelos de Comportamento de Jogadores; Telemetria; Game Analytics; Aprendizado de Máquina.

Table of contents

List of figures

List of tables

1
Introduction

The game market did not stop growing during the last decades [1] [2], with a special highlight for 2020, year in which even more people engaged on playing, due to the pandemic lockdown [3]. However, its getting even more difficult for developers to draw player's attention to their games, making them invest more time, money and effort on their projects, seeking for the most innovative features.

As stated by [4], "over the past decade and change, the number of video games on the market has increased exponentially. (...) For independent developers struggling for eyeballs against blockbuster mega-franchises like Assassin's Creed and Call of Duty, it's easy to feel like the deck is stacked against them from the start. But the exponential growth of indie games on Steam has tightened the vise against them even further, making it harder to stand out in an ever-crowding market."

Also in a research by [5], 37% of the games registered on Steam, the most accessed online game store today [6], have not even been uploaded once by registered users. And even though most of the bought games are uploaded, just a few of them can draw players attention in a significant way, being easy for the player to get bored or even frustrated with the experience [7].

This very competitive scenario makes the game industry consumers seek for the most addictive, high quality, and innovative games [4]. Thus, game designers need better strategies to create an attractive gameplay, and a way of creating a game with high *replay value* (i.e., a form to create a compelling video game experience that keeps players coming back multiple times).

One of the well-known strategies is to use game data to analyze player behavior, focusing on improving specific gameplay characteristics [8], attracting older and newer player types. This research strategy is directly related to *game analytics*, which is an area focused on data analysis and metrics in games [9].

Therefore, we propose to build a generic API, that allied with telemetry, will classify the player, based on the extended Bartle's model presented by Bicalho et al (2019) [10].

Thus, we will base ourselves on his work, considering some improvements that were already implemented, like: database to register each match attributes; attributes were adapted to percentage and mean values, allowing the data to not depend on the current level characteristics. Moreover, we will test different supervised and unsupervised machine learning algorithms, seeing which one can be used as the main one, and which ones can be used depending on the game genre.

To test the classification's accuracy we will use the same questionnaires from the 2019's work. The first was based on the work by Schneider et al. (2016) [11], which presents a questionnaire containing twenty questions, resulting in a percentage for each player type, with five weighted responses, differing from *Bartle Test of Gamer Psychology* [12] [13]. The second one followed the same

model, with weighted questions and answers, but basing on Adams' and Ip's work [13].

To test the accuracy of the different algorithms used, we implemented a test bench in Python, combining each supervised algorithm with each unsupervised algorithm, also varying what we called "Most Important Attributes Indexes" (MIAI). The end classification result also changes by picking different MIAIs, as they are used by an intermediate algorithm, that translates the cluster names (like "Cluster 0") to the equivalent player behavior model (like "Hardcore Achiever"). This results in a total of 132 MIAI combinations (as the game studied has 12 attributes and we choose a pair of MIAIs each iteration) with 24 algorithm combinations.

Finally, our work presents an Unity Plugin, that allied with an Python API, will allow its users to test 24 different approaches to obtain the player classification. It can be considered a generic approach, as it does not depend on the game genre. Moreover, all the algorithms run online, while the game is being executed, differentiating this approach from the ones that simply run based on the data collected.

This dissertation is organized as follows. The section *Related Works* presents the works that needed to be studied or read, with special focus on the ones that called themselves "generic" methods. The section *Theoretical Background* explains each machine learning algorithm used (supervised and unsupervised) and the Player Behavior Model chosen. *Methodlogy* presents how we applied our approach in the context of a Space Shooter game, also detailing the API and Plugin structure and the questionnaire design. *Questionnaire Application and Results* presents the results we obtained by running our test bench over the current logged data, focusing on the TOP 3 best algorithms, in relation to accuracy, the mean accuracy of the approaches combined, and other statistics. *Conclusion* presents the summary of our project and the future work.

2
Related Work

There were few works that called themselves "generic" in the field of player behavior or player classification. One of them was [14], which describes a generic method of interaction-based player classification, which consists of three components: (i) intercepting player interactions, (ii) finding player types through fuzzy cluster analysis and (iii) classification using Hidden Markov Models (HMM). Even though the steps applied are similar to ours, they did not use the best of the Fuzzy Logic, as interceptions between clusters are not considered. Also, there was no validation if the classification results were accurate or not, it was just a proof of concept.

Other work that used HMM to classify players was [15], comparing its efficiency in identifying player classifications, with Adaptive Memory Based Reasoning (AMBR). This method is "a variant of Memory Based Reasoning (MBR), based on the frequencies of player actions", but, as the paper concludes, it is not as efficient as HMM. Moreover, this work is very similar to our approach in the way they classify each player, using an adapted version of Bartle's Behavior Model, including: Killer, MarkovKiller, InexperiencedMarkovKiller and ExperiencedMarkovKiller.

These classification types are very similar to Bicalho's work in 2019 [10], which used an Extended version of Bartle's Player Models. This model added a new axis to identify what they called "Player Dedication", allowing a player to be classified not just as an Achiever, for instance, but as a Casual Achiever or a Hardcore Achiever. Our approach is an extension of this work, using the same methodology (involving clusterization and supervised learning), but expanding it to a more generic level, allowing to apply it to different game genres.

A more evolved use of graphs is done in [16]. They register the player actions, and other gameplay elements, as nodes in a Provenance Graph, allowing to understand which action lead to the current player state, what made he lose a life point, for instance. It is also very similar to [10], testing its efficiency in a Space Shooter game, and in other contexts. However, this work is limited to a proof of concept, as it lacks the accuracy test we do in our approach.

Also, there are some programs that already deal with data analysis and machine learning, like Orange [17] and KNIME [18]. Both have a friendly interface, that helps users to load the data and run any available algorithm, just by dragging and dropping the functionalities. Also, graphics from different types can be generated in what they call the Analytics Platform, and all data can come from a database. However, these programs just work offline, which makes our approach a better fit for the game context, as the main goal is to see player classification "on the fly", and update it during gameplay.

Unity also has an Analytics plugin, which helps improving games developed using this engine, and making a more compelling and engaging experience for the target audience [19]. "Games built with Unity can use Unity analytic tools to measure user experience and gameplay metrics, monitor an app's performance, and monetize the application." [20] Even though some events can

be created and measured, these statistics will be used more for a marketing purpose, to convert more players, than analysing player types. Also, there is no integration of these tools with unsupervised and supervised Machine Learning algorithms.

3
Theoretical Background

In this section, we will talk about the concepts that had to be studied to apply our following methodology. Among them, we have the machine learning algorithms applied (unsupervised and supervised), and the behavior models that were tested and validate throughout the experiments.

3.1
Unsupervised Algorithms

The chosen unsupervised algorithms are the ones that create the clusters based on a the $NClusters$ parameter. From the ones available in the *scikit-learn* Python library [21], K-means, Spectral Clustering, Agglomerative Clustering and Birch algorithms are the ones that follow that rule. In this section, we will explain how each one of these algorithms work.

3.1.1
K-means

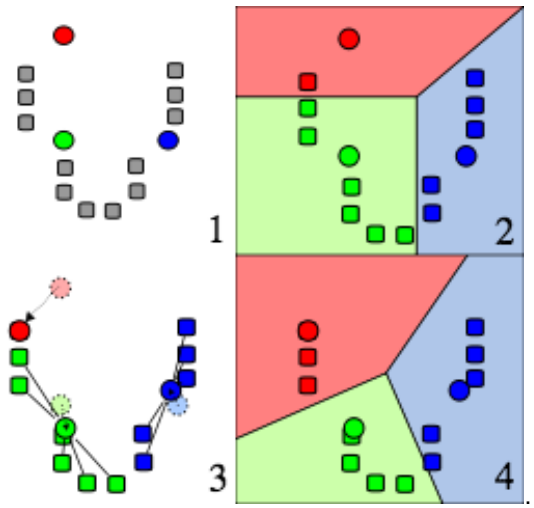

Figure 3.1: K-Means four main steps.

K-means is an unsupervised technique that only receives unlabeled training sets, making predictions from the attributes of each point [22]. This algorithm places each point in one of the K clusters, or groups, according to specific criteria. It works in three main steps [23]:

1. Choose K random centroids (points in the given attributes' domain) to represent the median of each cluster (Fig. 3.1, step 1);

2. Place each data point in the cluster with the nearest median, resulting in K separated clusters (Fig. 3.1, step 2), on the form of a Vornonoi diagram [24];

3. Each algorithm iteration runs through the whole data set, re-positioning its median based on the cluster values stored (Fig. 3.1, step 3). This re-positioning is repeated a determined number of times, resulting in clusters, like in the fourth step on Fig. 3.1, containing the points that are similar to each other.

3.1.2 Spectral Clustering

As its name points out, Spectral Clustering is an algorithm that bases itself on Linear Algebra's Spectral Theorem [25], as it involves several matrix manipulations. It is usually used to identify clusters like the ones show in Figure 3.2.

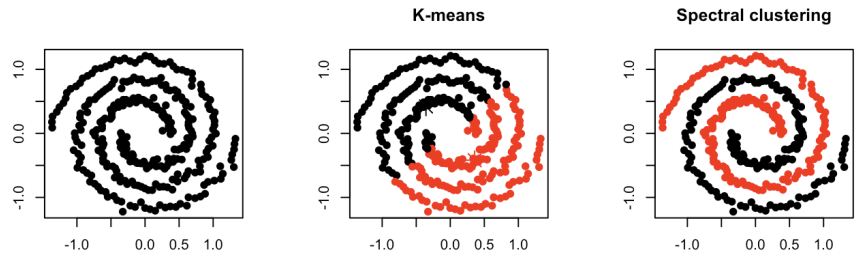

Figure 3.2: Spectral Clustering and K-Means comparison for a spiral data points.

While algorithms like K-means assume the format of clusters just based on distance values, Spectral Clustering uses the Kernel strategy to work with the data on different dimension, where they can be linearly separable, as shown in Figure 3.3. This type of technique is also used in algorithms like SVM, which we will talk about in other section.

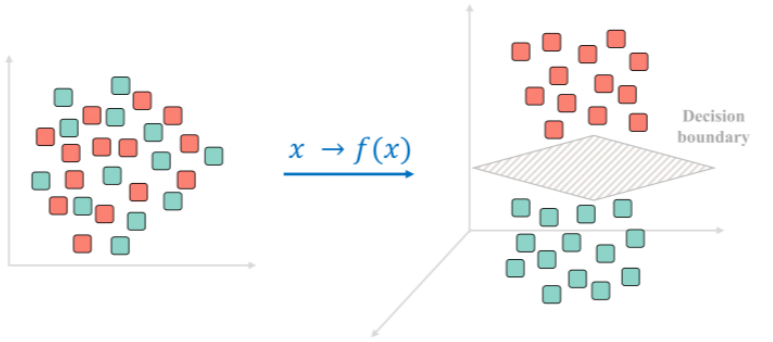

Figure 3.3: Example of Kernel function application.

The above explanation was adapted from [26] and [27].

3.1.3
Agglomerative Clustering

Hierarchical clustering, also known as hierarchical cluster analysis, is another used clustering algorithm. Different from K-means, which chooses the centroids of the clusters, it starts by treating each observation as a separate cluster. Then, it repeatedly executes the following two steps: (1) identify the two clusters that are closest together, and (2) merge the two most similar clusters. This iterative process continues until all the clusters are merged together (as we can see in Fig. 3.4). The main output of Hierarchical Clustering is a dendrogram, which shows the hierarchical relationship between the clusters, as show in Fig. 3.5.

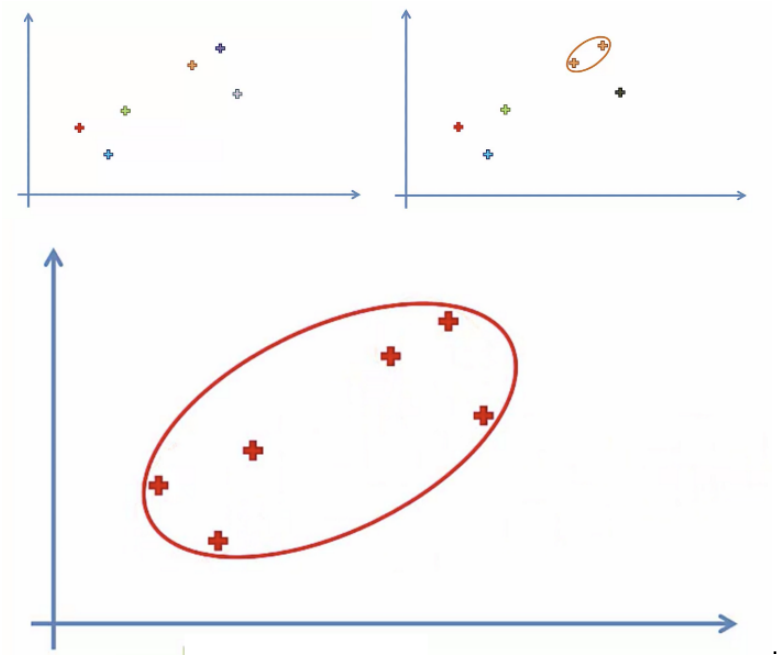

Figure 3.4: Example of Agglomerative Clustering steps.

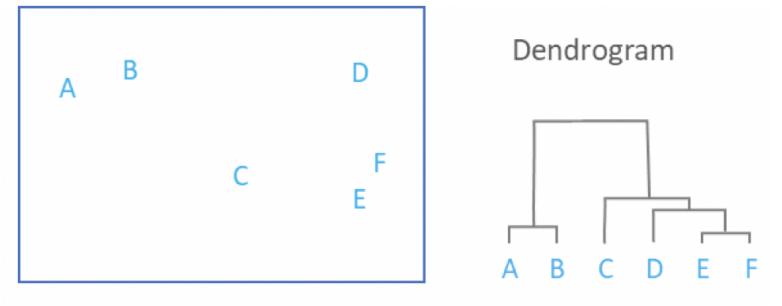

Figure 3.5: Example of Kernel function application.

The above explanation was adapted from [28] and [29].

3.1.4
Birch Algorithm

Clustering algorithms like K-means clustering do not perform clustering very efficiently and it is difficult to process large datasets with a limited amount of resources (like memory or a slower CPU). So, regular clustering algorithms do not scale well in terms of running time and quality as the size of the dataset increases. This is where BIRCH clustering comes in. Balanced Iterative Reducing and Clustering using Hierarchies (BIRCH) is a clustering algorithm that can cluster large datasets by first generating a small and compact summary of the large dataset that retains as much information as possible. This smaller summary is then clustered instead of clustering the larger dataset. BIRCH is often used to complement other clustering algorithms by creating a summary of the dataset that the other clustering algorithm can now use.

However, BIRCH has one major drawback – it can only process metric attributes. A metric attribute is any attribute whose values can be represented in Euclidean space i.e., no categorical attributes should be present. Before we implement BIRCH, we must understand two important terms: Clustering Feature (CF) and CF – Tree Clustering Feature (CF): BIRCH summarizes large datasets into smaller, dense regions called Clustering Feature (CF) entries.

Formally, a Clustering Feature entry is defined as an ordered triple, (N, LS, SS) where 'N' is the number of data points in the cluster, 'LS' is the linear sum of the data points and 'SS' is the squared sum of the data points in the cluster. It is possible for a CF entry to be composed of other CF entries.

The CF tree is the actual compact representation that we have been speaking of so far. A CF tree is a tree where each leaf node contains a sub-cluster. Every entry in a CF tree contains a pointer to a child node and a CF entry made up of the sum of CF entries in the child nodes. There is a maximum number of entries in each leaf node. This maximum number is called the threshold. We will learn more about what this threshold value is. Parameters of BIRCH Algorithm:

- threshold: threshold is the maximum number of data points a sub-cluster in the leaf node of the CF tree can hold.
- branching_factor: This parameter specifies the maximum number of CF sub-clusters in each node (internal node).
- n_clusters: The number of clusters to be returned after the entire BIRCH algorithm is complete i.e., number of clusters after the final clustering step. If set to None, the final clustering step is not performed and intermediate clusters are returned.

The above explanation was adapted from [30] and [31].

3.2 Supervised Algorithms

The chosen supervised algorithms are the ones available in the *scikit-learn* Python library, with them being Decision Tree, Naive Bayes and its variations, Support Vector Machines, Stochastic Gradient Descent, Neural Networks and Random Forest algorithms. In this section, we will explain how each one of these algorithms work (except for Neural Networks and Random Forest).

3.2.1 Decision Tree

Decision Tree is the most successful and one of the most straightforward learning algorithm, being easy to implement and serving as an excellent introduction to supervised learning [32]. We can consider it a supervised algorithm because it receives a set of labeled actions as training data and makes predictions for all unseen points [22]. In the context of our project, the labeled actions are the mapped player attributes during each gameplay session, while the prediction is the player classification based on the proposed taxonomy.

To explain the logic behind the algorithm, we will use a simple example, in which a tennis player should decide if he is going to play or not, based on the weather. Firstly, as we are dealing with machine learning, fourteen previous decisions, done in previous days, were recorded, as seen in Table 3.1. This table is considered as the training set for the decision tree.

Table 3.1: Decision tree example training set

Day	Outlook	Humidity	Wind	Play
1	Sunny	High	Weak	**No**
2	Sunny	High	Strong	**No**
3	Overcast	High	Weak	Yes
4	Rain	High	Weak	Yes
5	Rain	Normal	Weak	Yes
6	Rain	Normal	Strong	**No**
7	Overcast	Normal	Strong	Yes
8	Sunny	High	Weak	**No**
9	Sunny	Normal	Weak	Yes
10	Rain	Normal	Weak	Yes
11	Sunny	Normal	Strong	Yes
12	Overcast	High	Strong	Yes
13	Overcast	Normal	Weak	Yes
14	Rain	High	Strong	**No**

Looking at the data, we can see that there are three attributes, being *Outlook*, *Humidity* and *Wind*, and one goal predicate, being the decision to Play (Yes or No). But, to build the tree from Figure 3.7, we need to use a method to choose which attribute will be the first, i.e. which one of them is the most decisive.

To explain the logic behind the algorithm, firstly we need to use the concept of information entropy, which is defined as the value that quantifies uncertainty, i.e. the value of a choice [33]. The entropy value varies from zero to one, and is calculated using (3-1), in which: $E(S)$ is the current's situation (S) entropy; p_i is the probability of case i; n is the total number of cases; and b is the logarithmic base, representing the number of results.

As shown in Figure 3.6, the entropy value varies from zero to one, and is calculated using equation 3-1, in which: $E(S)$ is the current's situation (S) entropy; p_i is the probability of case i; n is the total number of cases; and b is the logarithmic base, representing the number of results.

$$E(S) = -\sum_{i=1}^{n} p_i \times log_b(p_i), E(S) \in [0, 1] \tag{3-1}$$

Applying this equation to the training set example, firstly we can notice there are 9 positive and 5 negative results, which implies in $b = 2$, $p_+ = \frac{9}{14}$ and $p_- = \frac{5}{14}$. Thus, the initial entropy (I) has a 0.94 value, according to 3-2.

$$\begin{gathered} E(I) = -p_+ \times log_2(p_+) - p_- \times log_2(p_-) \\ E(I) = -\frac{9}{14} \times log_2\left(\frac{9}{14}\right) - \frac{5}{14} \times log_2\left(\frac{5}{14}\right) \\ \boxed{E(I) = 0.94} \end{gathered} \tag{3-2}$$

To decide which of the attributes is the most relevant, we need to calculate their entropy gain towards the initial one, shown in (3-3). This calculation allows us to choose, comparatively, the attribute (A) that has the more significant entropy gain (G).

$$G(A) = E(I) - E(A) \tag{3-3}$$

But, before that, we need to calculate the entropy for each attribute value, multiplying for each probability. For instance, the entropy for Outlook (O) is 0.69, and 0.25 of gain, considering the entropy for Sunny (OS), Overcast (OO) and Rain (OR), as demonstrated in 3-4 and 3-5.

$$
\begin{aligned}
E(OS) &= -\frac{2}{5} \times log_2\left(\frac{2}{5}\right) - \frac{3}{5} \times log_2\left(\frac{5}{14}\right) = 0.97 \\
E(OO) &= -\frac{4}{4} \times log_2\left(\frac{4}{4}\right) - \frac{0}{4} \times log_2\left(\frac{0}{4}\right) = 0 \\
E(OR) &= -\frac{3}{5} \times log_2\left(\frac{3}{5}\right) - \frac{2}{5} \times log_2\left(\frac{5}{14}\right) = 0.97 \\
E(O) &= \frac{5}{14} \times E(OS) + \frac{4}{14} \times E(OO) + \frac{5}{14} \times E(OR) \\
&\boxed{E(O) = 0.69}
\end{aligned}
\tag{3-4}
$$

$$
\begin{aligned}
G(O) &= E(I) - E(O) \\
G(O) &= 0.94 - 0.69 \\
&\boxed{G(O) = 0.25}
\end{aligned}
\tag{3-5}
$$

Using this same method to each attribute, we obtained $G(H) = 0.15$ and $G(W) = 0.05$, approximately. This implies in $G(0)$ being chosen as the most decisive attribute, as its gain is bigger than the others. Finally, the entropy is recalculated for each value of the last chosen attribute, resulting in the tree shown in Fig. 3.7.

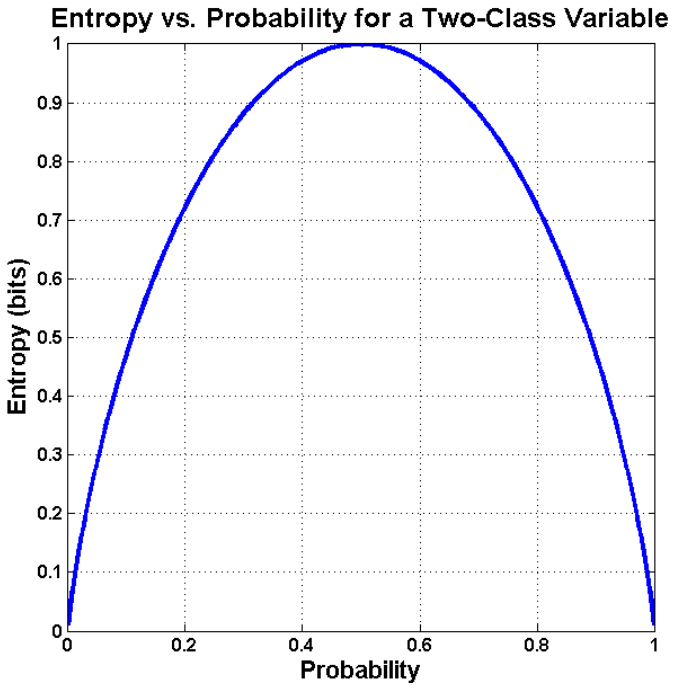

Figure 3.6: Graph from entropy function

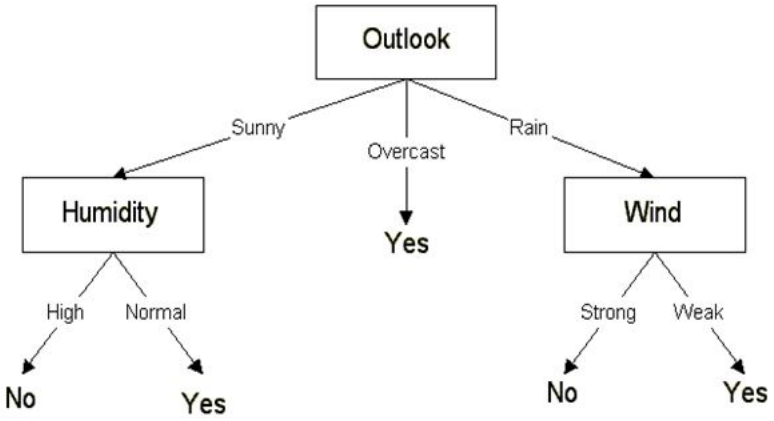

Figure 3.7: Decision tree example: Play Tennis

3.2.2 Naive Bayes

It is a classification technique based on Bayes' Theorem with an assumption of independence among predictors. In simple terms, a Naive Bayes classifier assumes that the presence of a particular feature in a class is unrelated to the presence of any other feature.

For example, a fruit may be considered to be an apple if it is red, round, and about 3 inches in diameter. Even if these features depend on each other or upon the existence of the other features, all of these properties independently contribute to the probability that this fruit is an apple and that is why it is known as 'Naive'.

Naive Bayes model is easy to build and particularly useful for very large data sets. Along with simplicity, Naive Bayes is known to outperform even highly sophisticated classification methods.

Bayes theorem provides a way of calculating posterior probability P(c|x) from P(c), P(x) and P(x|c). As we can see in (3-6): P(c|x) is the posterior probability of class (c, target) given predictor (x, attributes); P(c) is the prior probability of class; P(x|c) is the likelihood which is the probability of predictor given class; P(x) is the prior probability of predictor.

$$P(c|x) = \frac{P(x|c) \times P(c)}{P(x)} \tag{3-6}$$

To explain the logic behind the algorithm, we will use the tennis example again. Now that we already have the data, we need to follow the next steps:

- Convert the data set into a frequency table;
- Create likelihood table by finding the probabilities like Overcast $probability = 0.29$ and probability of playing is 0.64;

– Now, use Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

There are three types of Naive Bayes model under the scikit-learn library, which will be explained below. In our approach we just used Gaussian and Bernoulli.

– Gaussian: It is used in classification and it assumes that features follow a normal distribution;
– Multinomial: It is used for discrete counts. For example, let's say, we have a text classification problem. Here we can consider Bernoulli trials which is one step further and instead of "word occurring in the document", we have "count how often word occurs in the document", you can think of it as "number of times outcome number x_i is observed over the n trials".
– Bernoulli: The binomial model is useful if your feature vectors are binary (i.e. zeros and ones). One application would be text classification model where the 1s & 0s are "word occurs in the document" and "word does not occur in the document" respectively.

The above explanation was adapted from [34].

3.2.3 Support Vector Machines

Support Vector Machine (SVM) is a supervised machine learning algorithm that can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is a number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot). Support Vectors are simply the coordinates of individual observation. The SVM classifier is a frontier that best segregates the two classes (hyper-plane/ line).

Similarly to Spectral Clustering, it uses the kernel function, which takes low dimensional input space and transforms it to a higher dimensional space, i.e. converts not separable problem to separable problem, as we can see on the sequence shown in Fig. 3.8.

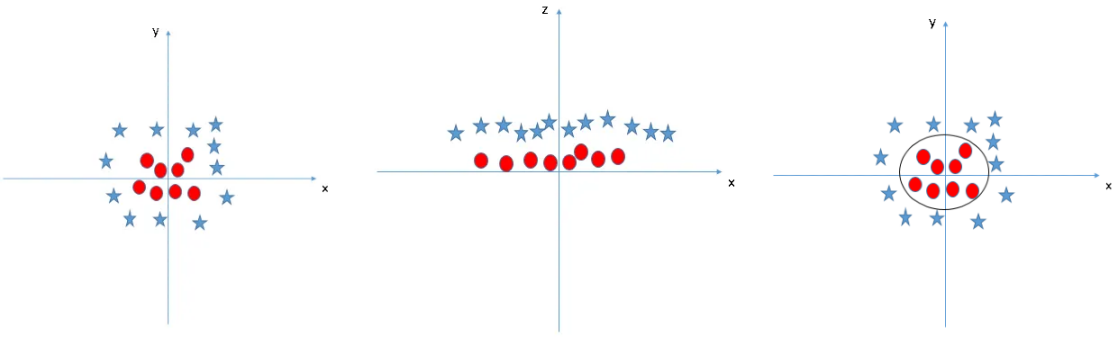

Figure 3.8: Example of SVG algorithm Kernel Function

The above explanation was adapted from [35].

3.2.4
Stochastic Gradient Descent

Let's start by understanding the Gradient Descent Algorithm, which is an iterative algorithm, that starts from a random point on a function and travels down its slope in steps until it reaches the lowest point of that function. Let's use the parabola shown in Fig. 3.9 as the main example. The steps of the algorithm are:

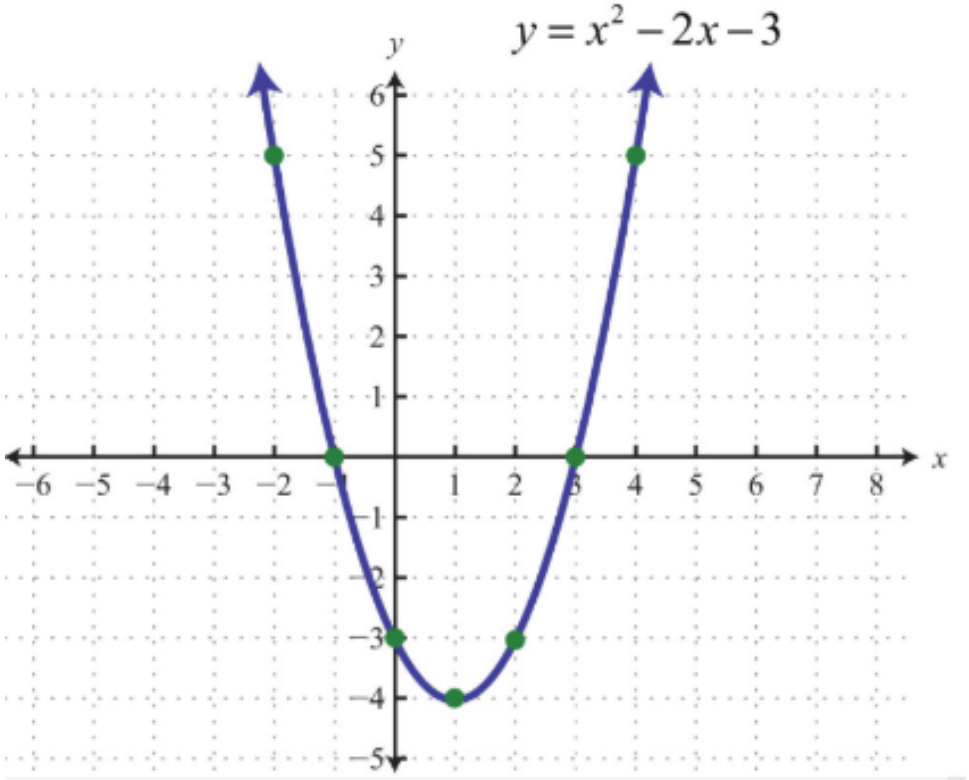

Figure 3.9: Example of parabola for the SGD algorithm

1. Find the slope of the objective function with respect to each parameter/feature. In other words, compute the gradient of the function.

2. Pick a random initial value for the parameters (To clarify, in the parabola example, differentiate "y" with respect to "x". If we had more features like x1, x2 etc., we take the partial derivative of "y" with respect to each of the features);

3. Update the gradient function by plugging in the parameter values;

4. Calculate the step sizes for each feature as : $stepsize = gradient \times learning\_rate$;

5. Calculate the new parameters as: $new\_params = old\_params - step\_size$;

6. Repeat steps 3 to 5 until gradient is almost 0.

The "learning rate" mentioned above is a flexible parameter which heavily influences the convergence of the algorithm. Larger learning rates make the algorithm take huge steps down the slope and it might jump across the minimum point thereby missing it. So, it is always good to stick to low learning rate such as 0.01. It can also be mathematically shown that gradient descent algorithm takes larger steps down the slope if the starting point is high above and takes baby steps as it reaches closer to the destination to be careful not to miss it and also be quick enough.

There are a few downsides of the gradient descent algorithm. We need to take a closer look at the amount of computation we make for each iteration of the algorithm. Say we have 10,000 data points and 10 features, with 1000 algorithm iterations. In effect we have 100 million computations to complete the algorithm. That is pretty much an overhead and hence gradient descent is slow on huge data. That is when the Stochastic version of the algorithms comes in hand, randomly picking one data point from the whole data set at each iteration to reduce the computations enormously.

The above explanation was adapted from [36].

3.3 Player Type Models

3.3.1 Bartle Taxonomy

In 1996, Richard Bartle created a taxonomy composed by four different types of players: *Achievers*, *Killers*, *Socializers*, and *Explorers* [37] (Fig. 3.10). Each one is defined as follows:

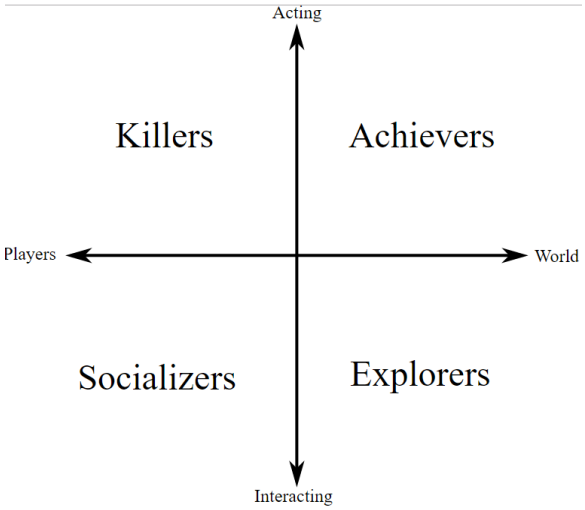

Figure 3.10: Bartle's Original Model

– *Achievers* are focused on mastering the game, on the rewards it has to offer. They share the world with other players, or non-playable characters (NPCs), and add a competitive element to the environment. Therefore, they are proud of their status in the game hierarchy, and how fast they reached their current level.
– *Killers* are focused on acting on other players, or NPCs, most of the time showing their superiority over them. They seek more power and abilities, that can help them affect others. Therefore, they are proud of their level of authority and their fighting skills.
– *Socializers* are focused on interacting and talking with other players, or NPCs. Also, finding more about other people is more interesting for socializers than competing, or bossing them. Therefore, they are proud of the relationships and of their influence towards other players.
– *Explorers* are focused on interacting with the world, the game environment. The sense of discovery or finding new areas and game elements fulfills them more than just achieving a great status in the game. Therefore, they are proud of their knowledge and of searching for new places and possibilities.

This taxonomy was made for MUD (Multi User Dungeons), which relates to current MMO RPGs, allowing to classify players based on their behavior during gameplay. However, how could this classes be applied to different types of game genres? Some singleplayer games doesn't have even fully interactive NPCs for the player to socialize with. For instance, in action shooter game, like the one we chose for this research, the only NPCs are the enemies' spaceships.

3.3.2 Bartle Extended Model

The solution was brought in 2019 in [10], presenting a extended model, including a Gamer Dedication axis, and showing that other genres could use the Bartle's Taxonomy by adapting it. In their example, they used just the upper quadrants, related to Killers and Achievers, as shown in Fig. 3.11. With the new axis, however, the model can be interpreted with a 2 axis simplification, with 4 classes remaining: Casual Killer, Casual Achiever, Hardcore Killer and Hardcore Achiever.

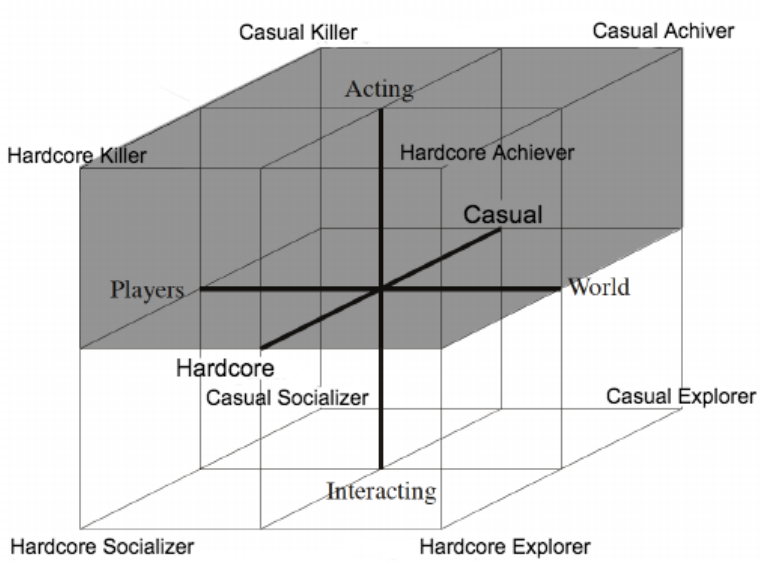

Figure 3.11: Bartle's Extended Model

In the 2019's work, we used these four types in a singleplayer *shoot'em up* game, the same we are using as a study case in this work, which gathers players' behavior attributes during each match. The right profile is chosen using K-means and Decision Tree algorithms, based on data from 138 previous gameplay sessions. This whole method was tested using two new questionnaires to match the player's profile evaluation with the game's final profile, revealing accuracy between 75% and 80%. More details can be found in [10].

This model is adaptable for any game genre with different types of players. For instance, if this model were applied to a game in which Socializers, Killers and Achievers are the most relevant classifications, the developers would use the first three quadrants, instead of just the first two. Therefore, the quadrants being used depends on the game genre and the developers approach when classifying player behavior. There is no single or "right" way to use this behavior model.

4
Methodology

Now that all the concepts needed were presented, we can use them to compose our methodology. Therefore, we divided this section in three subsections: *Game Mechanics*, which will present how the Space Shooter game works and how the data was collected during gameplay; *Player Classification*, which will present how we combine unsupervised and supervised techniques to obtain the current Player Behavior Archetype in real-time; and *API and Plugin Structure*, which will present how all the gameplay data is processed by the API and how this method can be edited by the game developer depending on his scenario.

4.1
Game Mechanics

To test the proposed generic method, we created a *shoot'em up* game, called *Space Shooter*, developed using one of the most used by the game industry, the Unity Game Engine [38]. We can also classify this game more generically as an action game, as its main challenge is to destroy all the innumerable hordes of different enemies that shoot in player's direction [39]. The game was uploaded to Itch.io, allowing players to play it from their browsers. Its latest version can be played by accessing the following link: *Space Shooter Game*. A screenshot of the game is shown in Fig. 4.1.

Figure 4.1: *Space Shooter* gameplay screenshot.

Before starting the gameplay, the player must insert their e-mail (for identification purposes) and choose between two spaceships: one that shoots lasers, and the other that shoots a spinning energy bullet, as seen in Figure X. After that, he enters the game match, starting with a "Placeholder" classification, as no input was given to our classification algorithm. He can move using "WASD" or the keyboard arrows, and shoot using the mouse left button or the "Space" keyboard button.

The gameplay is limited to one player at a time, also having just one stage to be completed. Each match is defined by 12 gameplay attributes (shown below), which are updated each 0.5 seconds.

1. Mean of direction changes ;
2. Mean of the position in X axis;
3. Mean of the position in Y axis;
4. Mean time in movement;
5. Total of items collected (*);
6. Number of coins collected (*);
7. Number of destroyed enemies (*);
8. Percentage of game completed;
9. Number of shots fired (*);
10. Number of accurate shots (on target/enemies) (*);
11. Number of inaccurate shots (that surpass the game boundaries) (*);
12. Total of shots taken.

The attributes marked with (*) are normalized based on, respectively: total number items spawned; total number of coins collected; and total number of enemies spawned. This normalizations allow us to make the data depend even less on the gameplay changes. For example, if the total game time is increased, more coins can appear if compared to the older version. If we kept the absolute values, the older matches would not be compatible with the new ones, invalidating hundreds of matches.

In the and, the last updated value of each attribute is converted into a JSON format, and saved as a new match on our MongoDB database, through an API made in Python language, agnostic to the game's code. This and the last paragraph changes were an improvement, if compared to previous works like [10], which persisted the data locally instead of posting to a cloud service.

4.2
Player Classification

With each previous match already registered in the database, they are all considered as part of the unsupervised training data. Therefore, we load this data to a matrix (each line is a match and each column is one of the 12 attributes listed in the last section), which will be passed to the unsupervised algorithm function, with the number of clusters chosen by the game developer through the Plugin Class. After the algorithm is executed, the resulting centroids are registered in an array, but without the archetypes labels yet.

For instance, if we run the Decision Tree algorithm in the dataset labeled with names from "Cluster 0" to "Cluster 4", the result tree would be similar to the one in Fig. 4.2. According to it, the most relevant attribute is the 8^{th} one (Number of Shots Fired). In comparison to the 2019's version, this one ignores the percentage of enemies destroyed as a relevant parameter to decide the final classification. The reason why the algorithm discarded the other attributes can be related to the change from Mean to Percentage in some values. The ideal would be to register all the statistics related to each attribute (total, mean, percentage and standard deviation) and test which one would fit the best with each algorithm.

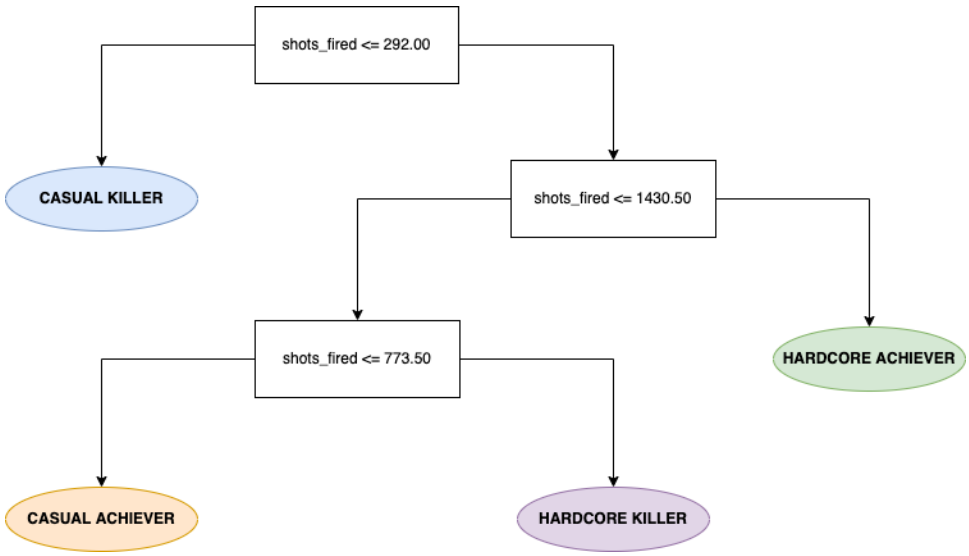

Figure 4.2: An Example of Decision Tree generated.

However, how can we pair the clusters with the player models? We created an algorithm that processes the cluster centroids and pair them to the cluster names, according to the two most important attributes. For instance, in the context of the Space Shooter project, we used the attributes 7 and 4, respectively, as initially they were the ones we mapped as most importants from the previous decision tree (Fig. 4.3). The algorithm finds out which centroid maximized the value of the most relevant attribute (7), labeling it as the first cluster name chosen (in the example, will be a "Hardcore Achiever"). The second centroid with the second biggest attribute value will be labeled as a "Hardcore Killer". The two other clusters will be labeled following the same logic, but considering the value of the second most important attribute (4).

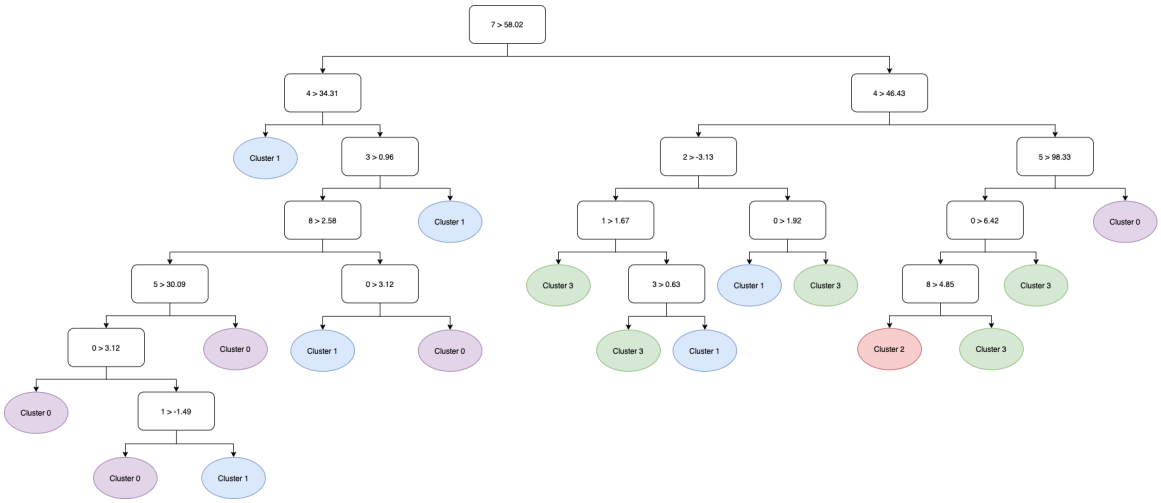

Figure 4.3: Decision Tree generated before cleaning the database.

This approach reproduces the human behavior of an expert in these game archetypes, allowing the game developer to just worry about the most important values (which he/she can decide based on a decision tree or on his expertise). Now with all the dataset labeled properly, we can run a supervised algorithm based on it. This algorithm will decide, each 5 seconds, what is the player's current classification, based on the current values of the attributes during the gameplay.

In the end, the final classification is based on the last time the supervised algorithm was executed, as the data from the whole match was gathered. The final archetype is shown to the player on the Game Over or the Victory screens, as shown on Fig. 4.4 and Fig. 4.5.

Figure 4.4: Example of *Space Shooter's* game over screen.

Figure 4.5: Example of *Space Shooter's* level complete screen.

4.3 API and Plugin Structure

There were some improvements from the 2019's project, mainly on where these algorithms run, and how much control the user has over the classification process. This control is done by values that are passed to the *Classifier Plugin* inspector parameters, which are divided in two groups: Classifier Parameters and Database Parameters. This script should be linked to the "GameController" class, if the game developer has one, or the class that controls most part of the game's main logic.

The first group parameters are: Cluster Num (Total Number of Clusters), Unsupervised Alg (Unsupervised Algorithm Name), Supervised Alg (Supervised Algorithm Name), Cluster Names (Cluster Name Array) and Most Important Attributes Indexes, as shown in Fig. 4.6. Each one is explained in the list below:

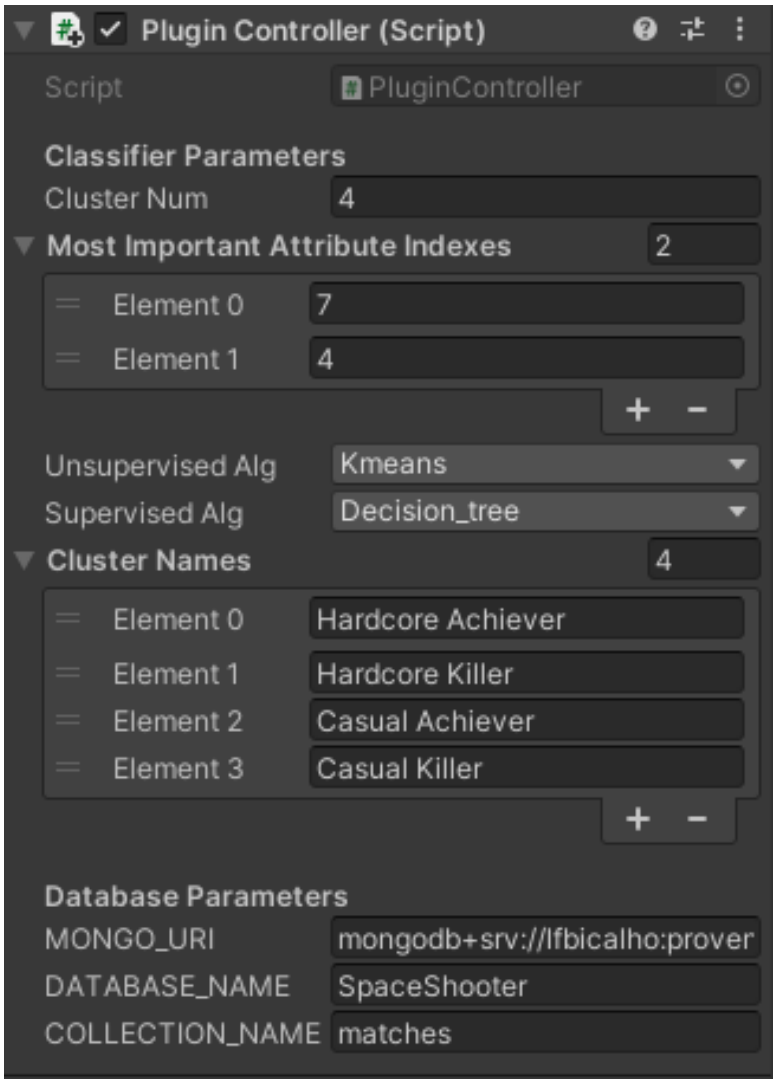

Figure 4.6: Using the Classifier Plugin in the Space Shooter Project.

1. Total Number of Clusters: The developer can choose it depending on the classification model chosen;

2. Unsupervised Algorithm Name: The developer can choose between the four algorithms explained in the *Theoretical Background* section. All of them are based on the number of clusters and generate an array of centroids;

3. Supervised Algorithm Name: The developer can choose between the seven algorithms explained in the *Theoretical Background* section;

4. Cluster Names Array: The developer can base its classification method in an already known Player Classification Model, like Bartle's, or use a Custom approach. If he/she chooses a custom approach, he/she must enter each archetype name, according to the total number of clusters;

5. Most Important Attributes Indexes (MIAI): As told before, the developer must choose the most appropriate attributes, from the gameplay attributes list, to be used for classification's labeling step.

The second group parameters are: MONGO_URI (Mongo Database URI), DATABASE_NAME (Created Database Name) and COLLECTION_NAME (Session Collection Name), as shown in Fig. 4.6. Each one is explained in the list below:

1. Mongo Database URI: The developer that wants to use our Plugin must create an account at this link. After that a database must be created. After these steps, an option named "Connect" will appear. This option will allow them to see the Mongo URI, as shown on Fig. 4.7, and pass this as a parameter to the Plugin Controller;

2. Created Database Name: The developer must create a database and pass its name to this parameter;

3. Session Collection Name: The developer must create a collection to store game sessions, independent of how the Session model was edited in Unity.

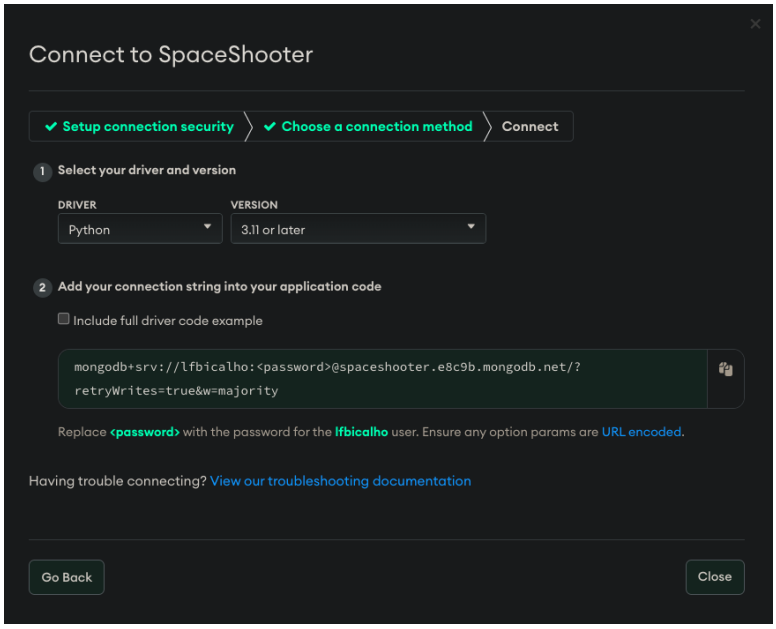

Figure 4.7: Retrieving MongoDB URI through their platform.

This makes the Plugin Classifier class, extremely generic, allowing the developer to have more control over the classification process, and even which steps are involved in it. Also, to avoid problems in the classifier execution, there are default values for each of the listed parameters.

However, other classes should be edited to allow the online classification to work, more specifically: Event, Telemetry and Session. In the first, the Types enum should be edited adding the attributes that define the developer's game session (an example of the Types used in the current case studied can be seen in Figure 4.8). In the second, the SetupEvents must be edited, using the function "CreateEvent" based on the created types and on the chosen statistics (an example used in the current case studied can be seen in Figure 4.9). In the last one, the attribute names shown in Figure 4.10 should be changed to the ones used in the Types enum.

```
public class Event
{
    public enum Type
    {
        directionChange,
        xPosition,
        yPosition,
        timeInMovement,
        item,
        coin,
        destroyedEnemies,
        levelCovered,
        shotsFired,
        accurateShots,
        inaccurateShots,
        shotsTaken
    }
```

Figure 4.8: Example of Type enum used in the Space Shooter game.

```
public static void SetupEvents()
{
    events.Clear();

    CreateEvent(Event.Type.directionChange, "mean");
    CreateEvent(Event.Type.xPosition, "mean");
    CreateEvent(Event.Type.yPosition, "mean");
    CreateEvent(Event.Type.timeInMovement, "mean");
    CreateEvent(Event.Type.item, "percentage");
    CreateEvent(Event.Type.coin, "percentage");
    CreateEvent(Event.Type.destroyedEnemies, "percentage");
    CreateEvent(Event.Type.levelCovered, "percentage");
    CreateEvent(Event.Type.shotsFired);
    CreateEvent(Event.Type.accurateShots, "percentage");
    CreateEvent(Event.Type.inaccurateShots, "percentage");
    CreateEvent(Event.Type.shotsTaken);
}
```

Figure 4.9: Setting up the main events for the Space Shooter game.

```
public class Session: IEnumerable<double>
{
    public string _id;
    public double firstAttribute;
    public double secondAttribute;

    public IEnumerator<double> GetEnumerator()
    {
        yield return firstAttribute;
        yield return secondAttribute;
    }

    IEnumerator IEnumerable.GetEnumerator()
    {
        return GetEnumerator();
    }
}
```

Figure 4.10: Session model example.

After the values are set, and the game executable is generated, the players can be properly classified during their matches. The ML algorithms, however, are not running inside Unity anymore. A Python API is constantly running on Heroku Server, available in the following link: `https://shooter-provenance-api.herokuapp.com/`.

4.3.1 Available API Routes

The API routes used in this project are divided in two categories: the ones related to classification and prediction (machine learning), and the ones responsible for counting all matches/sessions saved on the database and create new ones.

The first used route is from the second group and is accessed through the *"/setup-db"* URL (POST request). It is used to setup the database parameters, based on what the user passed to the Plugin Controller class.

After this setup, the second route is executed through the *"/classifier"* URL (POST request). It is part of the first group, and is used to load the classifier based on the already registered entries in the MongoDB database. In this process, the unsupervised algorithm is executed, the clusters are labeled based on the MIAI values, and the supervised classifier is set and ready to receive the gameplay attributes values.

Before the player reaches the selection menu, a GET request is sent to the second route, part of the second group, through the *"/count-sessions"* URL. This route is responsible for counting all the matches/sessions, showing how many players have passed through that experience. After this information is loaded, the selection menu opens, allowing the player to choose which spaceship he/she prefers.

After this selection, the game is initialized and the gameplay data starts to be collected and sent to the API, through a POST request to the *"/class"* URL, sending the match/session data (attributes) as its body. This route is part of the first group, running at each defined interval (usually 500 milliseconds, set through the Telemetry class, which will be explained in the next subsection). It returns the current classification based on the classifier decision.

At the end of the match, a POST request to the *"/sessions"* URL is sent, registering a new entry on the sessions/matches collection (the attributes are used as the request body). This route is part of the second group and allows the dataset to grow organically, making the final classification even more accurate for the next players.

Besides these main routes, there are auxiliar ones, mostly related to Sessions' CRUD (Create, Read, Update and Delete) actions, which are better explained in the API documentation, available in this following link: API Github Documentation. Moreover, there is a special route, accessed through the "/sessions" URL (DELETE method). Game developers can send requests to it to clean the database, as some unwanted data could have been registered during the development process or during tests, allowing for a clearer dataset.

4.3.2 Plugin's Classes and Models

As our main deliverable is a Unity Package that accesses the API, described in the previous section, we need to understand the classes that make all this logic work. There are two main Classes, three classes related to the "Telemetry" group and four "Models". Each group will be defined and their classes will be explained.

- Main Classes
  - "ClassifierController", a singleton responsible for receiving all the configuration data and storing it in variables. It is also responsible for calling some routines implemented by the "ClassificationAPI";
  - "ClassificationAPI", a class responsible for defining all functions that make requests to the API, and for formatting all the request bodies.
- Telemetry Classes
  - "Telemetry", a singleton responsible for implementing functions related to setting up, creating and updating Events;
  - "Event", each event is related to a Session attribute. For instance, the player taking a shot from the enemy can be considered an Event. Each one has four types of statistics associated with it: total, mean, standard deviation and percentage. The developer must choose which one should be the "main statistic", i.e. the the one that will be used to decide player's classification. This is done in the "SetupEvents" function, from the "Telemetry" class, as shown in Fig. 4.9;

  - "InitialData", which is responsible for populating the database with initial data that could just be saved to text files or CSVs. This allows developers to not loose early data that is still relevant for the classification process. The only condition is to have each attribute value separated by comma.

- Model Classes
  - "Session", is the model that defines each attribute that represents the game session/match, as shown in a hypothetical example in Fig. 4.10;
  - "ApiResponse", is the model that defines the ApiResponse format for the first group of routes, which return an array of Sessions, a message and a boolean to tell if it was a successful request (this two last ones also appear in the next models);
  - "ApiCountResponse", is the model that defines the ApiResponse format for the "/count-sessions" URL, returning an integer as main value;
  - "ApiTreeResponse", is the model that defines the ApiResponse format related to the second group of routes, which return a string related to the Classifier prediction.

4.4 Questionnaire Design

We decided to use the same two questionnaires that were used in the 2019's work [10], to test if the classification shown at the end of the completed game is compatible with the player's profile. The first verifies if the player is classified as an Achiever or a Killer. It was based on the work by Schneider et al. (2016) [11], which presents a questionnaire containing twenty questions, resulting in a percentage for each player type.

Their approach differs from the usual *Bartle Test of Gamer Psychology* [12] [13], as it does not have binary questions forcing the player to fit in a profile (e.g. one answer indicates an achiever profile and the other a socializer one). They use, instead, the same five answers for every question:

- "I do not understand/I do not identify myself" (0 points);
- "I identify myself a little" (1 point);
- "I identify myself partially" (2 points);
- "I identify myself" (3 points);
- "I identify myself totally" (4 points).

Each answer has a weight related to it, making it more difficult to have different people choosing the same one. This approach is very similar to the Likert scale, as shown by Joshi et al. [40]. Moreover, the player who does not identify him/herself with any answer scores 0% in every profile, which is a more honest and precise evaluation [11].

The questions are also different from the usual Bartle's Test, having five questions to identify each player type (total of 20). We only use ten of them, as

we considered Achievers and Killers only. The following list shows the proposed questions [11]:

- Achiever
  - "I like to conquer new badges in games";
  - "I get impressed with players that conquered high rewards";
  - "I play electronic games until the end with 100% of achievements";
  - "I love new items and medals";
  - "I like exposing my achievements (for example, on Facebook)".
- Killer
  - "I am very competitive in games";
  - "I like exploding things in games";
  - "My favorite games are first person shooters";
  - "I am known for my aggressiveness in games";
  - "I do not like talking in games, what I really like is shooting".

To decide whether the player is an Achiever or a Killer, we decided to sum the points related to the questions of each archetype, and get the maximum value from their result, as shown in (4-1). If the sum result is equal for both types, the player is classified as both, lowing the chances of the game classification being wrong. This also happens, for instance, if the player is defined as 55% Killer and 45% Achiever, i.e. he/she is classified as both if the distance between both Killer and Achiever percentage is below or equal to 10 percentage points.

$$PT = \max\left(\sum_{i=1}^{5} A_i, \sum_{j=1}^{5} A_j\right), (A_i, A_j) \in [0, 4] \tag{4-1}$$

The second questionnaire focuses on identifying if the game user is a Casual or a Hardcore player. To measure his/her dedication, we used the previous cited definition of a hardcore player, on the fifteen characteristics presented in Section III-D. Thus, we created the following questions (associated with each characteristic, respectively):

- "I always deal with technology and seek for new releases and trends" (7 points);
- "I like to have the latest high-end computers/consoles" (7 points);
- "I'm willing to pay anything for a game" (5 points);
- "I prefer violent/action games" (1 points);
- "I prefer games that have depth and complexity" (3 points);
- "I play games over many long sessions" (10 points);
- "I always search for the game industry latest information" (6 points);

– “I frequently talk about games, both via social media and with people” (10 points);
– “I always feel happy when completing (or defeating) a game” (7 points);
– “I don’t get easily frustrated while playing a game” (9 points);
– “I am usually engaged in competition with myself, the game, and other players” (6 points);
– “I started playing games when I was little” (2 points);
– “I have played all the types of game genres, and I constantly compare one game to another” (10 points);
– “I buy games and consoles on their pre-release, or import them from other countries to be one of the first to play” (9 points);
– “I think of modifying and extending some of the games I play” (8 points);

To answer each question, we repeated the same method used in the first questionnaire, with those five weighted responses. Besides that, we can notice that each question is also weighted, as we based ourselves on the work by Adams and Ip [13]. This method allows us to give more importance to some questions, when compared to others.

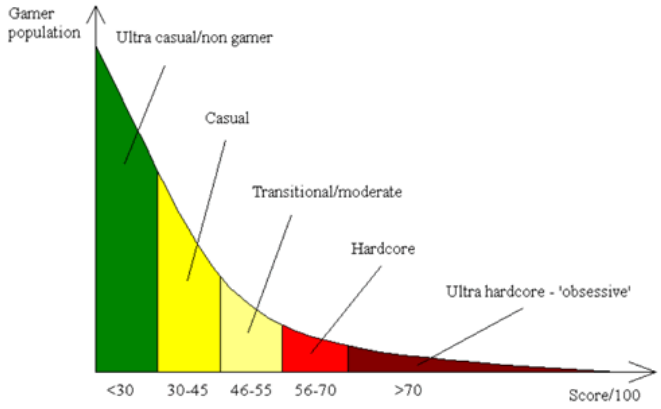

Figure 4.11: Casual and Core by gamer dedication.

To quantify the player dedication, we used (4-2), in which: A_i represents the answer weight for question i; Q_i represents the weight for question i; and GD is the gamer dedication factor, which is represented by the sum of the multiplication of both question and answer weights, divided by 4 multiplied by the weights, representing the maximum points the user can make. This results in a percentage, that is interpreted according to Fig. 4.11, as shown in the list below, considering Non-gamers as Casuals, and Ultra Hardcore gamers as Hardcores.

1. Casual gamer - Has GD factor below or equal to 45%;
2. Moderate gamer - Has GD factor between 45% and 55%, with these limits included;
3. Hardcore gamer - Has GD factor above 55%.

$$GD = \frac{\sum_{i=1}^{15} A_i \times Q^i}{\sum_{i=1}^{15} 5 \times Q^i} \quad (4\text{-}2)$$

These questionnaire models are available through the *this link*. Their application not just allows our project to stand out in comparison to other work on the same area, but also as a model for game developers to test the method accuracy in their context.

5
Questionnaire Application and Results

Our MongoDB Database currently has 135 entries, each one representing a different match/game session. We registered all the 12 attributes, as detailed in previous sections, also saving the final classification generated by the API.

To test each combination of algorithms and MIAIs, we designed a test bench, which generated a CSV and a Bar Plot for each MIAI pair, considering the 12 attributes and their combination with the other 11 (as the order matters, it ends up being a total of 132 different combinations). As we can see, we don't have the Spectral Clustering column, as we had some issues with the algorithm during the tests. Also, we used five different versions of K-means, which will be explained below [41] [42]:

- kmeans++: Selects initial cluster centers for k-mean clustering in a smart way to speed up convergence;
- kmeans-rand: Choose *n_clusters* observations (rows) at random from data for the initial centroids;
- kmeans++ (elkan): The last two versions apply, by default, the "Lloyd" K-means algorithm. The "elkan" variation can be more efficient on some datasets with well-defined clusters, by using the triangle inequality. However it's more memory intensive due to the allocation of an extra array of shape (n_samples, n_clusters);
- kmeans-rand (elkan): Combination of random initializer with the "elkan" algorithm;
- mb-kmeans: "Mini Batch KMeans is a variant of the KMeans algorithm which uses mini-batches to reduce the computation time, while still attempting to optimise the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence time of k-means, mini-batch k-means produces results that are generally only slightly worse than the standard algorithm.

  The algorithm iterates between two major steps, similar to vanilla k-means. In the first step, samples are drawn randomly from the dataset, to form a mini-batch. These are then assigned to the nearest centroid. In the second step, the centroids are updated. In contrast to k-means, this is done on a per-sample basis. For each sample in the mini-batch, the assigned centroid is updated by taking the streaming average of the sample and all previous samples assigned to that centroid. This has the effect of decreasing the rate of change for a centroid over time. These steps are performed until convergence or a predetermined number of iterations is reached.

  MiniBatchKMeans converges faster than KMeans, but the quality of the results is reduced. In practice this difference in quality can be quite small", as shown in Fig 5.1.

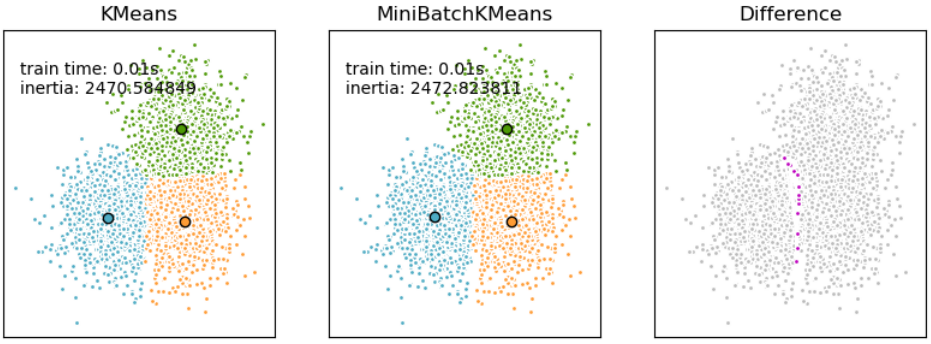

Figure 5.1: Mini Batch K-means and K-means comparison.

Considering that there are two classification dimensions (as shown in Figure 3.11): one to classify accordingly to Bartle Axis; and the other to classify accordingly to Gamer Dedication Axis. Therefore, we have two distinct situations when comparing the classification results from the questionnaire with the ones from the algorithms: the first happens when the accuracy is 100% if the player classification found by the API was exactly the same from the questionnaire, i.e. the API got both Bartle and Player Dedication classifications right, and we call it complete result; the second, on the other hand, will show 100% if the player classification found by the API was almost the same from the questionnaire, i.e. the API got at least one of the Bartle and Player Dedication classifications right, and we call it complete-partial result.

These data is registered on CSV files, saved into the “csvs" folder, with two other folders inside: complete and complete-partial. An example of the CSV content is shown in Tables 5.1 and 5.2. Each table is related to a different MIAI pair, showing the accuracy for each possible combination of supervised and unsupervised algorithms.

Table 5.1: Table generated for MIAI (3-9) (part 1)

	agglomerative	birch	kmeans++	kmeans-rand
decision_tree	60.00%	60.00%	56.00%	56.00%
gaussian_naive_bayes	60.00%	60.00%	56.00%	56.00%
bernoulli_naive_bayes	64.00%	64.00%	60.00%	60.00%
SGD	52.00%	52.00%	56.00%	60.00%
SVM	60.00%	60.00%	56.00%	56.00%
random_forest	60.00%	60.00%	56.00%	56.00%
neural_network	60.00%	52.00%	56.00%	**68.00%**

The questionnaire was applied to 50 players (25 after some clean-ups), to check the current method accuracy, as some changes were made since its application in 2019. On the first application, the result was an accuracy of almost 80%, which is quite considerable. This time the mean accuracy, considering the highest accuracy for each MIAI combination, is 62%, with a standard deviation of approximately 3.7; the maximum accuracy found for “complete” classifications was 72%, and the minimum was 44%. However, if we look at the mean accuracy found for “complete-partial” classifications, we get 98%, with a standard deviation of approximately 4.1; the maximum accuracy,

Table 5.2: Table generated for MIAI (3-9) (part 2)

	kmeans++ (elkan)	kmeans-rand (elkan)	mb-kmeans
decision_tree	56.00%	56.00%	56.00%
gaussian_naive_bayes	56.00%	56.00%	56.00%
bernoulli_naive_bayes	60.00%	60.00%	60.00%
SGD	60.00%	56.00%	56.00%
SVM	56.00%	56.00%	56.00%
random_forest	56.00%	56.00%	56.00%
neural_network	52.00%	56.00%	56.00%

for this case, was 100% and the minimum was 76%. This means that the algorithm is capable of getting at least one of the two classifications, Bartle or Gamer Dedication, right, which is a step further in comparison to 2019's results. As some algorithms have a random start or not always get the same result, these

The combination that was more consistent in its results was Agglomerative/Hierarchical Clustering allied with Bernoulli Naive Bayes. And the top 3 combinations with the best results (72% complete accuracy, and 100% partial accuracy) were:

1. K-means++(elkan) + Neural Networks, considering MIAI=(1, 2);
2. Agglomerative Clustering + Neural Networks, considering MIAI=(1, 6);
3. Mini Batch K-means + Stochastic Gradient Descent, considering MIAI=(10, 0).

The reason why these exact combinations gave the best results, and why Partial Classification had a greater accuracy if compared to Complete Classification must be further investigated. The data and models developed so far does not allow us to explain the above mentioned results.

One thing that bothered us towards the development of this project was the nature of the classification results. While the classifications generated each 5 seconds, and at the end of the game session, are dynamic, i.e. can change depending on countless factors (player's physical and/or emotional stability, or not having played a game from that genre yet), the classification that we can get from the questionnaire is generic/global, allowing it to be applied in multiple contexts.

Comparing these two results can be either satisfactory or disappointing, depending on the generic result, as a player can be considered Hardcore, as he plays lots of different games, from different genres, but he is not that good in the game genre used in the case study, which can make him be classified as Casual. The same with the Casual player, as he maybe is not that into games, but liked to play the current game genre, which can make him be classified as Hardcore.

These situations are really hard to map, but seemed quite relevant to the context of our project. For future work, we suggest the application of questionnaires that identify Killers, Achievers, Casual, and Hardcore, or any classification desired, more precisely to the game genre being studied.

6
Conclusion

Our system presents the accuracy of each combination of player model, unsupervised algorithm, and supervised algorithm and allows the game developer to test all these possibilities online, that is, while the game is running. This specificity makes this work the only one in the literature that combines different models and algorithms online, giving a satisfying and accurate result at the end of the process. Also, because we implemented the system as a plugin, the user can only link it to the game in the Unity Engine and associate its classes with the proper entities to make it work properly.

More than a generic approach, the proposed system allows us to test different combinations of unsupervised and supervised machine learning algorithms, which will run before each match and during them to identify the profiles based on the current inputs. This process will make the player profile identification process easier to be applied to the game, allowing designers and developers who do not have too much experience with machine learning to understand the process and contribute to its results.

Finally, our work presents a Unity Plugin that, allied with a Python API, will allow its users to test 3696 different approaches - from the combinations of 4 different unsupervised algorithms, 7 different supervised algorithms and 132 MIAIs pairs - to obtain the player classification. It does not substitute Unity Analytics, as it generates different types of information, working more as a compliment than a competitor.

Our proposal represents a generic approach, as it does not depend on the game genre. The clustering algorithms run online, differentiating this approach from those based on the data collected in a previous closed session. Our system makes the classification process easier for game developers, who can test all these possibilities in a safe environment and during all game production phases, from the prototype to the final product.

However, there is still much work to do. Firstly, the database should have support to not just register one statistic per attribute, but all the four (total, mean, percentage and standard deviation) statistics available in the plugin. With this implemented, more tests could be done, comparing which one is more appropriate for the current game. Also, the used test bench could be provided as an endpoint of the Python API, allowing the game developer himself/herself to test the combinations and map the best result.

As the current approach was tested with only one game, from a specific genre, it would be necessary to test it in other contexts, with other types of games. This would prove the effectiveness of the algorithm, and reinforce the "generic" side of the approach.

The questionnaires were applied to just 25 players and there are just 135 entries in the database. This numbers could be increased by applying this questionnaires to more people, and generating new entries in the database, by playing new matches or even populating with some generated "fake data". Also, other types of questionnaires - maybe some made specifically for the type

of game genre being tested - , with a less generic result, could be applied and compared to the algorithms results.

Other change that could be made in our approach is related to the intersections between Hardcore/Casual and Killer/Achiever. We suggest applying Fuzzy logic to get the probability of a single player being Hardcore, Casual, Killer and Achiever, respectively. This would enrich our data and make it easier to compare the algorithm and questionnaire results.

Even though we are combining the most relevant machine learning algorithms, it would be a great addition to test unsupervised algorithms that doesn't generate cluster centroids. This would require the MIAI algorithm to be adapted, but could generate better results in comparison to the ones we got with our current approach. Also, we could test the different approaches for the already implemented algorithms, like Agglomerative Clustering, that has 4 different versions, that could give different results.

7
Bibliography

[1] WIJMAN, T.. **Newzoo global games market report 2019**. Technical report, Newzoo, 2019.

[2] MIDDELHOFF, D.; SCHUNK, D.. **Gaming industry - facts, figures and trends**. Technical report, Clairfield International, 2018.

[3] NIELSEN. **3, 2, 1 go! video gaming is at an all-time high during covid-19**. https://www.nielsen.com/us/en/insights/article/2020/3-2-1-go-video-gaming-is-at-an-all-time-high-during-covid-19/, 2020. Acessed: 15/12/2020.

[4] WRIGHT, S.. **There are too many video games. what now?** https://www.polygon.com/2018/9/28/17911372/there-are-too-many-video-games-what-now-indiepocalypse, 2018. Acessed: 10/07/2019.

[5] ORLAND, K.. **Introducing steam gauge: Ars reveals steam's most popular games**. https://arstechnica.com/gaming/2014/04/introducing-steam-gauge-ars-reveals-steams-most-popular-games/, 2014. Acessed: 03/07/2019.

[6] PRESCOTT, S.. **The most popular desktop gaming clients, ranked**. https://www.pcgamer.com/the-most-popular-desktop-gaming-clients-ranked/, 2019. Acessed: 17/12/2020.

[7] LOVATO, N.. **16 reasons why players are leaving your game**. https://www.gamasutra.com/blogs/NathanLovato/20150408/240663/16_Reasons_Why_Players_Are_Leaving_Your_Game.php, 2015. Acessed: 15/12/2020.

[8] STATT, N.. **How artificial intelligence will revolutionize the way video games are developed and played**. https://www.theverge.com/2019/3/6/18222203/video-game-ai-future-procedural-generation-deep-learning, 2019. Acessed: 24/05/2019.

[9] EL-NASR, M. S.; DRACHEN, A. ; CANOSSA, A.. **Game Analytics: Maximizing the Value of Player Data**. Springer Publishing Company, Incorporated, 2013.

[10] BICALHO, L. F.; BAFFA, A. ; FEIJÓ, B.. **A game analytics model to identify player profiles in singleplayer games**. p. 11–20, 10 2019.

[11] SCHNEIDER, M. O.; MORIYA, T. U.; VIEIRA DA SILVA, A. ; NÉTO, J. C.. **Analysis of player profiles in electronic games applying bartle's taxonomy**. 09 2016.

[12] MULLIGAN, J.; PATROVSKY, B. ; KOSTER, R.. **Developing online games: An insider's guide**. 01 2003.

[13] BARR, M.. **The bartle test of gamer psychology**. https://matthewbarr.co.uk/bartle/, 2017. Acessed: 23/06/2019.

[13] ADAMS, E.; IP, B.. **From casual to core: A statistical mechanism for studying gamer dedication**. https://www.gamasutra.com/view/feature/131397/from_casual_to_core_a_statistical_.php, 2002. Acessed: 20/05/2019.

[14] ETHEREDGE, M.; LOPES, R. ; BIDARRA, R.. **A generic method for classification of player behavior**. p. 2–8, 10 2013.

[15] MATSUMOTO, Y.; THAWONMAS, R.. **Mmog player classification using hidden markov models**. p. 429–434, 09 2004.

[16] COSTA KOHWALTER, T.; GRESTA PAULINO MURTA, L. ; WALTER GONZALEZ CLUA, E.. **Capturing game telemetry with provenance**. In: 2017 16TH BRAZILIAN SYMPOSIUM ON COMPUTER GAMES AND DIGITAL ENTERTAINMENT (SBGAMES), p. 66–75, 2017.

[17] **Orange: Data mining**. https://orangedatamining.com/, 2022. Acessed: 30/05/2022.

[18] **Knime**. https://www.knime.com/, 2022. Acessed: 30/05/2022.

[19] **Unity analytics**. https://unity.com/products/unity-analytics, 2022. Acessed: 12/07/2022.

[20] G2. **7 unity analytics tools to track your app in 2020**. https://www.g2.com/articles/unity-analytics, 2017. Acessed: 12/07/2022.

[21] **Scikit learn**. https://scikit-learn.org/stable/, 2022. Acessed: 12/07/2022.

[22] MOHRI, M.; ROSTAMIZADEH, A. ; TALWALKAR, A.. **Foundations of Machine Learning**. The MIT Press, 2012.

[23] MACKAY, D. J. C.. **Information Theory, Inference & Learning Algorithms**. Cambridge University Press, New York, NY, USA, 2002.

[24] AURENHAMMER, F.. **Voronoi diagrams - a survey of a fundamental geometric data structure**. ACM Comput. Surv., 23(3):345–405, Sept. 1991.

[25] LEMAŃCZYK, M.. **Spectral Theory of Dynamical Systems**, p. 8554–8575. Springer New York, New York, NY, 2009.

[26] LUXBURG, U.. **A tutorial on spectral clustering**. Statistics and Computing, 17:395–416, 01 2004.

[27] KEERTHANA, V.. **What, why and how of spectral clustering!** https://www.analyticsvidhya.com/blog/2021/05/what-why-and-how-of-spectral-clustering/, 2021. Acessed: 27/06/2022.

[28] MAKLIN, C.. **Hierarchical agglomerative clustering algorithm example in python**. https://towardsdatascience.com/machine-learning-algorithms-part-12-hierarchical-agglomerative-clustering-examp 2018. Acessed: 26/06/2022.

[29] BOCK, T.. **What is hierarchical clustering?** https://www.displayr.com/what-is-hierarchical-clustering/, 2022. Acessed: 28/06/2022.

[30] ZHANG, T.; RAMAKRISHNAN, R. ; LIVNY, M.. **Birch: An efficient data clustering method for very large databases**. In: PROCEEDINGS OF THE 1996 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, SIGMOD '96, p. 103–114, New York, NY, USA, 1996. Association for Computing Machinery.

[31] GEEKS, G. F.. **Ml | birch clustering**. https://www.geeksforgeeks.org/ml-birch-clustering/, 2022. Acessed: 28/06/2022.

[32] RUSSELL, S.; NORVIG, P.. **Artificial Intelligence: A Modern Approach**. Prentice Hall Press, Upper Saddle River, NJ, USA, 3rd edition, 2009.

[33] SHANNON, C. E.. **A mathematical theory of communication**. SIGMOBILE Mob. Comput. Commun. Rev., 5(1):3–55, Jan. 2001.

[34] RAY, S.. **6 easy steps to learn naive bayes algorithm with codes in python and r**. https://www.analyticsvidhya.com/blog/2017/09/naive-bayes-explained/, 2021. Acessed: 27/06/2022.

[35] RAY, S.. **Understanding support vector machine(svm) algorithm from examples (along with code)**. https://www.analyticsvidhya.com/blog/2017/09/understaing-support-vector-machine-example-code/, 2021. Acessed: 27/06/2022.

[36] SRINIVASAN, A. V.. **Understanding support vector machine(svm) algorithm from examples (along with code)**. https://towardsdatascience.com/stochastic-gradient-descent-clearly-explained-53d239905d31, 2019. Acessed: 27/06/2022.

[37] BARTLE, R.. **Hearts, clubs, diamonds, spades: Players who suit muds**. 06 1996.

[38] DEALS, T.. **This engine is dominating the gaming industry right now**. https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/, 2016. Acessed: 15/05/2019.

[39] ADAMS, E.. **Fundamentals of Game Design**. New Riders Publishing, Thousand Oaks, CA, USA, 3rd edition, 2014.

[40] JOSHI, A.; KALE, S.; CHANDEL, S. ; PAL, D.. **Likert scale: Explored and explained**. British Journal of Applied Science & Technology, 7:396–403, 01 2015.

[41] SCULLEY, D.. **Web-scale k-means clustering**. p. 1177–1178, 01 2010.

[42] ARTHUR, D.; VASSILVITSKII, S.. **K-means++: The advantages of careful seeding**. volumen 8, p. 1027–1035, 01 2007.