University of Pristina "Hasan Prishtina"

Topic: Applying the Bisecting K-Means clustering algorithm to cluster heroes in the video game Dota 2

Mentors: Prof. Dr. Eng. Lule Ahmedi

Candidate: Nora Ibrahimi

Pristina, December 2019

Abstract

IntelliDota is a project implemented using the Scala and Flutter programming languages, the combination of which brings a smart, stable, fast and helpful application, through which we can analyze data sources, visualize and apply different metrics and algorithms on them. This project is divided into two parts, this document is about the IntelliDota Clustering part.

IntelliDota Clustering is the second part of the IntelliDota project, which uses a data set obtained from Kaggle. The data set consists of 500,000 rows and 12 columns, and the Bisecting K-Means algorithm, which is a version of K-Means, is applied to this data. The data set is in CSV format, which means comma separated *values* .

Before applying the algorithm, we first need to pre-process the relevant data set. Since we did not need to fix errors, such as missing columns, etc., the only pre-processing that needs to be performed is the grouping of values in the hero_id column. This column represents the hero's identification number, and if we find the average for each hero in this grouping, then we get some representative values that are symbolic of the respective hero. With this pre-processing, we derive a final data set consisting of 110 rows and 12 columns. 110 rows since there are 110 heroes in the game.

After this phase, we apply the Bisecting K-Means algorithm and save the trained model so that if we want to perform any other real-time clustering, we do not have to retrain the model but use the ready-made model.

Since this report contains the second part of the IntelliDota project, it is known that there is a Docker image that is not published on any Cloud. This image will be published on Google Cloud Platform via the Google SDK, a set of tools that help us perform operations via a CLI. Once the image is published, we have a public web that we can access at any time.

With the ability to access services at any time, an application has been created that visualizes the respective representations for both data sets, be they metrics, algorithm phases that have passed, correlation matrices or many other details. The main part of this application is the ability to perform prediction or clustering in real time, where the result is returned in a very short time depending on the data set that we have selected. This has been implemented through Flutter, a tool built by *Google for building native* applications . for smartphones, screens or browsers. Allows for interface development (eng. *interface*) in a fast, expressive and flexible way. The programming language is *Dart* .

Table of figures

Table of tables

Table of contents

1 Entrance

Just as human curiosity never dies, so does the advancement of technology. Although it is not wrong to say that technology has almost reached its peak, there are still unexplored paths that we can say represent a technological world in itself. Among them, and among the most popular and sought after by people is artificial *intelligence* .

1.1 motivation

Artificial intelligence, also referred to as machine *intelligence* It is about stimulating the intelligence acquired and developed by machines [1].

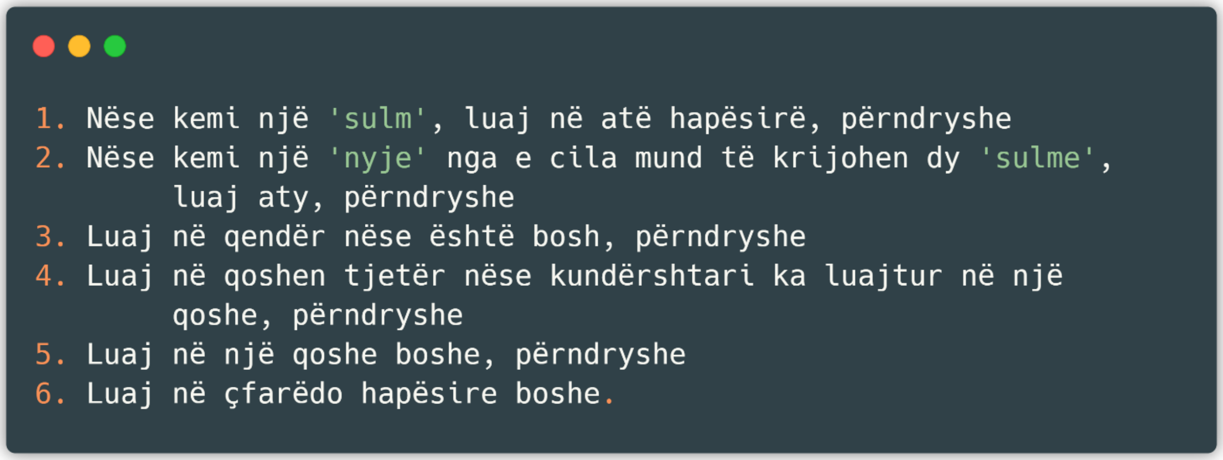
This science is otherwise defined as the field of study of intelligent agents. (eng. *intelligent agents*) *:* a device that understands its environment and takes actions that maximize the likelihood of achieving its goal.

We say that an agent is rational if it does the right thing, that is, every action in certain circumstances is correct. But how can we define what is correct and what is not? If the agent goes through a sequence of actions and states that satisfy us, we say that the agent has had a good performance [2].

So, machines are trying to adopt human understanding skills. and problem solving (eng. *learning and problem solving*) *.*

The goal of an intelligent agent can be simple, such as playing a game of GO, or complex, such as performing mathematical operations.

The basic principle of artificial intelligence is the use of algorithms. Algorithms are a set of instructions that a computer can execute. A complex algorithm is built as a collection of simple algorithms.



```
1. Nëse kemi një 'sulm', luaj në atë hapësirë, përndryshe
2. Nëse kemi një 'nyje' nga e cila mund të krijohen dy 'sulme',
        luaj aty, përndryshe
3. Luaj në qendër nëse është bosh, përndryshe
4. Luaj në qoshen tjetër nëse kundërshtari ka luajtur në një
        qoshe, përndryshe
5. Luaj në një qoshe boshe, përndryshe
6. Luaj në çfarëdo hapësire boshe.
```

Fig 1: Algorithm of a tic-tac-toe game

Some algorithms are capable of learning from a set of data, as in our case, where the algorithm learns a strategy or a good way (*rule of thumb*) which it applies to new data, and some other algorithms can write other algorithms themselves.

Some of *the* learning algorithms, such as *nearest - neighbor* , *decision trees ,* or *Bayesian* networks can theoretically approximate any data to a given function (if they have infinite memory and time). Artificial intelligence has been stimulated since ancient times, and among the most popular approaches are:

- Symbolism, otherwise known as formal logic: if the person has a cold, then they have the flu.

- Interference Bayesian (*Bayesian interference*) *:* if a person has a cold, then there is a probability that they also have the flu.

- *Support* Vector Machine or *Nearest-neighbor:* after reviewing the data of people who have a cold, including age, symptoms, and other factors, and these factors belong to the current patient, we say that the patient has the flu.

A machine is said to be intelligent if it passes the Turing test, a test that was originally designed to provide a satisfactory definition of intelligence. The test is passed if it is impossible to tell whether the answers come from a human or a machine after a series of human questions [3].

1.2 Problem Description

It is known that today we find applications of artificial intelligence in almost every area of life, especially in the field of computing, starting from recommender systems, prediction systems, and many others.

Considering these applications, we have attempted to build an assistant for the popular video game Dota 2 as it would not only help people during the game, but would also enrich the community with a unique data set (since we are creating one ourselves through the relevant calls) and source code that presents the progress of the project and an example of how to achieve something similar both in terms of artificial intelligence, and in terms of making calls and filtering the Json structure.

It should be noted that there are many datasets related to this topic on the internet, but due to the techniques and methodologies we want to implement, we have chosen to create one ourselves.

Our problem can be described in several stages, the most important of which are:

- Data and Algorithm – this phase contains the main steps related to machine learning. Starting from obtaining data, cleaning it, translating attributes to be readable for the machine, and up to the possibility of using the algorithm on the processed data set.

- Organization and Presentation in Flutter – this phase involves communicating with the calls created in the Scala part and defining a design that is suitable and responsive for mobile devices while still being readable for the user. Finally, the logical connection of the view to the data and testing on different devices followed.

So, we say that our application has successfully completed its work if it is able to classify and cluster the data we give it as input in real time, quickly and in an organized manner.

The intelligent part of the application can be implemented in many ways, starting from the Python programming language with the relevant libraries that enable the creation of an application similar to what we want to do, to writing an algorithm on our own. But since it was not considered reasonable to 'reinvent the wheel', we have decided to use the Scala programming language together with Spark, which offers a powerful arsenal for manipulating data and artificial intelligence. A more detailed description of why we chose this language with the relevant libraries will be explained later.

As for the data creation part, there are other ways (such as using a programming language to create the calls) but not methodologies, which means that the data set we have created can only be obtained through the official Steam website through the relevant calls and filters.

2 Artificial Intelligence – Clustering as Unsupervised Learning

In general, there are different forms of machine learning, but three more specifically represent almost every form of learning. We have:

- Unsupervised learning – the agent learns ways to solve or compute a given problem based on input even though there is no feedback from any given source (i.e., the algorithm is neither rewarded nor punished for the chosen action). Essentially, unsupervised learning is synonymous with clustering. The learning process is unsupervised because the input examples are not labeled as classes. We use clustering to discover classes within the data. For example, an unsupervised learning method might take as input a set of images of handwritten numbers. Suppose it finds 10 clusters of data. These clusters might correspond to 10 distinct numbers from 0 to 9, respectively. However, since the training data is unlabeled, the learned model cannot tell us the semantic meaning of the clusters found.

- Reinforcement learning – the agent learns from a series of output actions, whether they are punishing or rewarding actions. For example, a tip at the end of a taxi ride tells the algorithm that it has acted well, otherwise it has acted badly. It is up to the algorithm to decide which parts of itself have made it perform the action well.

- Supervised learning – the agent learns from input-output pairs, and learns a function by which it creates a scheme that orients future input values to output values. In principle, this type of learning is synonymous with classification. Supervision during learning comes from labeled examples in the training set. For example, in the postal code recognition problem, a set of handwritten postal codes and the corresponding machine-readable translations are used as training examples, which supervise the learning of the classification model.

In this application, clustering will be used precisely as an unsupervised learning method, a technique that will be described below precisely with the relevant algorithms.

2.1 Unsupervised learning algorithms

Clustering is a technique in artificial intelligence that involves grouping data, so we can use clustering to classify data into certain groups. In theory, data in the same group should have similar characteristics and qualities, while data in different groups should have distinctive attributes [4].

Clustering, on the other hand, is an unsupervised learning technique and is widely used for statistics and data analysis in various fields.

In our application, we have used K-Means as a clustering technique, but we will also show the other widely used technique Mean-Shift. from which there is no need to determine the number of central points, otherwise called centroids .

2.1.1 K-Means

It is probably the most widely used and widely used algorithm for clustering. This is because it is easy to use and understand. We are describing the algorithm in steps to explain how it works.

- First, we choose a number of clusters and randomly place their center points. To find the number we want to use, it is good practice to look at the data and see 'with the naked eye' if they are similar. Center points are also vectors like any other actual point.

- Each data point is classified by calculating the distance between the current point and each set of center points, and the data point is classified into the set whose distance is smallest among all the sets of center points.

- Based on these classification points, we recalculate the center points by taking the average value of all center groups.

- We repeat these steps for a certain number of times or until the cluster centers do not change much. Another option is to randomly initialize the cluster centers several times and choose the result that performed best.

K-Means has the main advantage in speed, since all it does is calculate the distances between points or cluster centers; very small calculations. Therefore it has linear complexity O(n) [5].

An example of the K-Means algorithm looks like this (Fig 2):

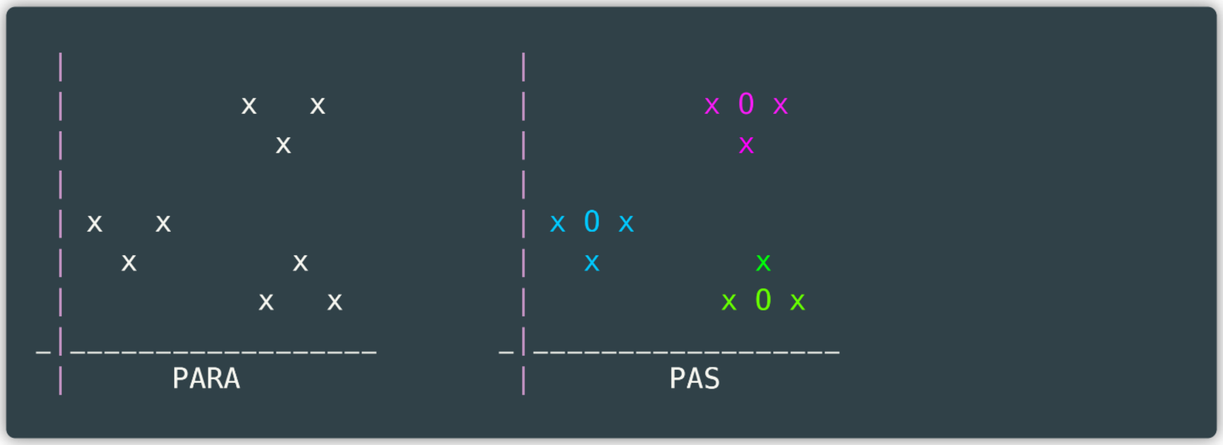


Fig 2: K-Means visually

where the first graph presents the data before clustering, i.e. ordinary data without any grouping, while the second graph presents the data grouped based on color, where 3 central points marked with 0 are seen. (assuming we have chosen 3 central points, so k = 3). And if a new data item arrives, it is grouped in the cluster to which the distance is closest.

On the other hand, there are some drawbacks, starting with the fact that you have to manually set the number of clusters of center points (variable k). This is not a good thing because we want to extract sound statistics from the data set and it would be nice if the algorithm took care of this part.

In addition, as mentioned above, K-Means assigns cluster centers randomly, which is why the results are different in each environment where the algorithm is run. So, the results are not reusable and we lack consistency. Other methods are more stable.

2.1.2 Mean Shift

Mean Shift is a parameterless clustering procedure, famous in the field of computer vision and image processing, whose output, unlike the K-Means algorithm, does not depend on explicit assumptions about the shape of the point distribution, the number of clusters, or any form of random initialization [6].

Mean Shift builds on the concept of kernel density estimation . *density estimation (KDE*). KDE is a method for estimating the underlying distribution, also called the probability density function, for a data set. It works by placing a kernel (Kernel *)* at each point in the data set. A kernel is a mathematical expression for a weighting function. There are different types of these kernels, but the most famous is the Gaussian kernel . By summing all the individual kernels a probability surface (i.e. *the density function*) will be generated.

Mean Shift exploits this idea of KDE, imagining what would happen if the points were to climb up to the nearest peak on the KDE surface. This continues until all the points have reached the peak and these points will eventually merge into a series of points, near the local maxima of the distribution. Those points that converge to the same local maxima are considered members of the same cluster, the number of which depends on the bandwidth of the KDE surface, so if we took a very small value we would result in as many clusters as there are points in the data set, and otherwise we would have only one cluster formed.

2.1.3 Bisecting K-Means

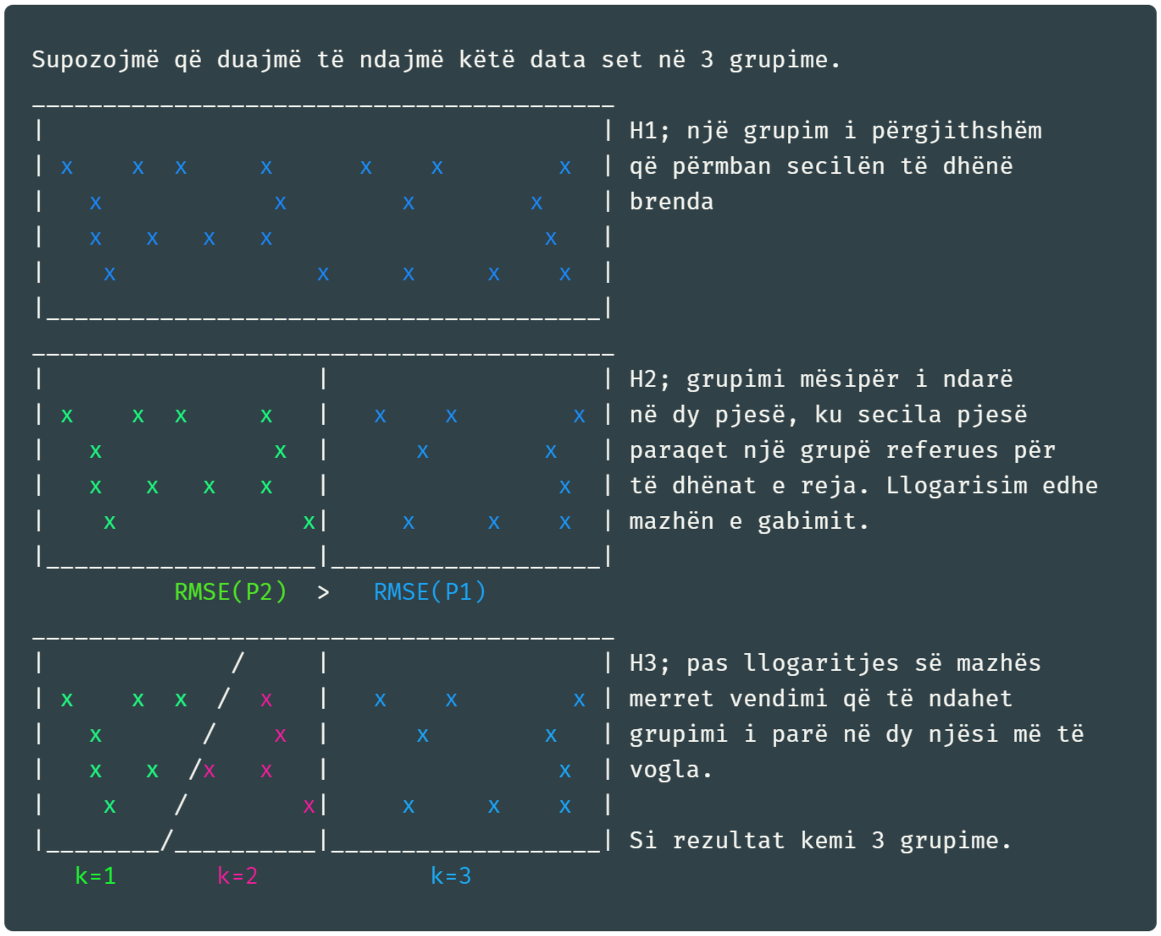
The idea behind this algorithm is almost the same as that of the K-Means algorithm, but it differs in certain details.

Unlike the K-Means algorithm, this algorithm does not select random points to then measure the distance, but divides our sample in half, continuing with the part with the largest margin of error until K clusters are formed [7].

We describe below the exact steps that this algorithm takes:

- It collects data into a single group, meaning that all the data in the data set is located in a single cluster.
- K-Means with a value of 2 is applied, so our cluster is divided into two relatively equal parts, these parts determined by the algorithm itself.
- The margin of error is calculated according to the RMSE metric and the data set with the largest margin of error is selected to be divided again into smaller groups.
- The previous step is repeated until the number of clusters is equal to K.

An example of what this looks like graphically is shown below (Fig 3).



```
Supozojmë që duajmë të ndajmë këtë data set në 3 grupime.

-----------------------------------------
|                                       | H1; një grupim i përgjithshëm
| x    x  x     x      x    x        x  | që përmban secilën të dhënë
|   x            x        x        x    | brenda
|   x   x   x   x                   x   |
|    x              x     x     x    x  |
|_______________________________________|

-----------------------------------------
|                   |                   | H2; grupimi mësipër i ndarë
| x    x  x     x   |   x    x        x | në dy pjesë, ku secila pjesë
|   x            x  |      x        x   | paraqet një grupë referues për
|   x   x   x   x   |                x  | të dhënat e reja. Llogarisim edhe
|    x             x|     x     x    x  | mazhën e gabimit.
|___________________|___________________|
          RMSE(P2)  >   RMSE(P1)

-----------------------------------------
|             /     |                   | H3; pas llogaritjes së mazhës
| x    x  x  /  x   |   x    x        x | merret vendimi që të ndahet
|   x       /    x  |      x        x   | grupimi i parë në dy njësi më të
|   x   x  /x   x   |                x  | vogla.
|    x    /        x|     x     x    x  |
|________/__________|___________________| Si rezultat kemi 3 grupime.
   k=1       k=2           k=3
```

Fig 3: Bisecting K-Means visually

Among the many clustering algorithms, we have chosen, as mentioned above, the Bisecting K-Means algorithm. Among the main reasons why we have chosen this algorithm over the others is that it has high accuracy, the way it works suits our data set and that the phenomenon of randomness is less present, not to mention the fact that it is among the newest algorithms introduced in Spark.

3 Introduction to the project

Having described artificial intelligence and the main clustering algorithms as representatives of unsupervised machine learning, we can now move on to describing the solution provided to the problem, as well as the results obtained.

We are starting with a description of the software tools that were used, so that their use can be understood later, depending on the case.

- *IntelliJ* – build tools (eng. *Integrated Development Kit* , (IDEA) in which the application is written, built (eng. *build*) AND TESTED (eng. *test*). This tool is written in the Java programming language and was chosen among many others for its high organization, the many possibilities it offers, and the ready-made packages for both Scala and Play.

- *Postman* - which is currently one of the most popular tools used in API testing, where API stands for Application Programming Interface. *Application Programming Interface*) and allows software applications to communicate with each other through API calls.

- *Android Studio* - is the official integrated development environment for Google's Android operating system, designed specifically for Android development. This environment was used for developing the Flutter application, as it offers great convenience in coding and is also the recommended environment by Flutter.

- *Powershell* – the tool used to manage the Docker image. Unlike other CLIs (Command Line Interfaces), this tool is more functional, feature-rich, and has a more user-friendly interface.

- *Docker* – the tool used to create the image so we can publish our services on the internet accessible to everyone.

- *Git* – the platform on which the work progress is coordinated. Since this platform dominates the work market, it has been seen as reasonable to practice this way of working even more. It works through the command line interface (*Command Line Interface, CLI*).

- *Github* – the platform where *Git* puts source files. This platform generally collects all public projects written by every programmer in the world and offers them to researchers so that they can access both source code and applications that are built as utilities, such as the tool used below *Ngrok.*

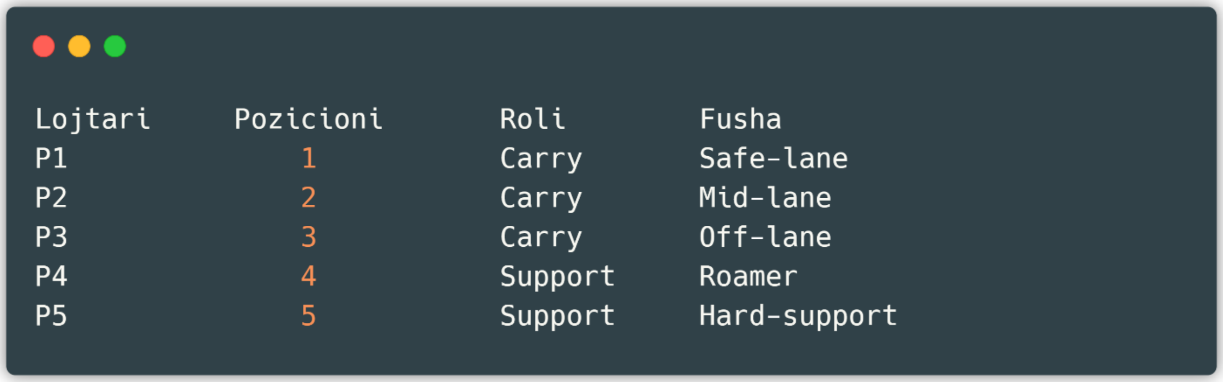
IntelliDota is an application built in Scala and Flutter that attempts to *cluster* and *classify* the *results of a video game* , in our case the video game *Dota 2.* We will call the part of the application that performs the classification IntelliDota Classification and the part of the application that performs the clustering IntelliDota Clustering. Before we continue further with the application, we will give a brief description of what this video game is and what we want to preach.

3.1 Computer game Dota 2

Dota 2 is one of the MOBA type games that means real-time multiplayer online battle arena (MOTA), also known as real-time action strategy, which with the refining measures applied over the years has reached a high level of sophistication and with this has attracted not only many players, both casual and professional, but has also become part of numerous studies and various experiments, especially in the field of Artificial Intelligence.

Dota 2 stands for Defense *of the Ancients* , where players choose from a pool of over a hundred heroes, forming two teams consisting of five players each, occupying separate spaces on the map. Each of the ten players independently controls a character, called a hero, with unique abilities and different play styles. During the match, the player and his team accumulate experience points and gold to purchase items for their heroes in order to penetrate the twelve towers of the opposing team. A team wins by being the first to destroy the enemy's main base called *the "Ancient". Dota 2* has a special emphasis on tactics and team coordination, and a heavy focus on the strategy of building up forces as quickly as possible and choosing the order of upgrading the hero's spells [8].

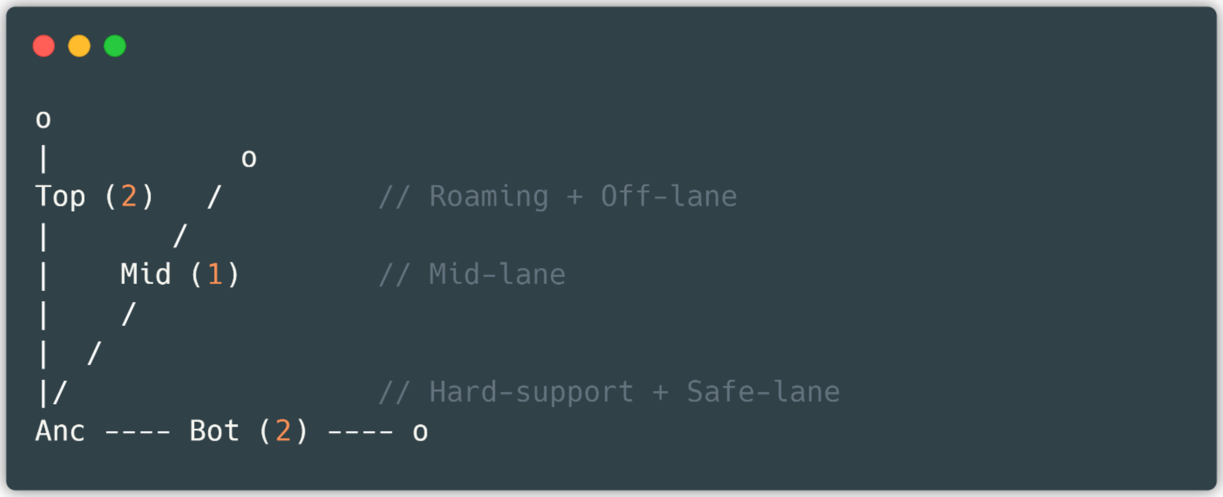
Teams in the game have a higher level division, namely carries *and* supports . Essentially , each *support supports* a *carry* (Fig 4*)* .



```
Lojtari     Pozicioni       Roli        Fusha
P1              1           Carry       Safe-lane
P2              2           Carry       Mid-lane
P3              3           Carry       Off-lane
P4              4           Support     Roamer
P5              5           Support     Hard-support
```

Fig 4: Formation of Dota 2 video game players

While from a high perspective, the team is organized in such a formation (Fig 5):



```
o
|          o
Top (2)   /         // Roaming + Off-lane
|       /
|    Mid (1)        // Mid-lane
|    /
|  /
|/                  // Hard-support + Safe-lane
Anc ---- Bot (2) ---- o
```

Fig 5: Team organization

Each player, regardless of role, has hundreds of statistics, but among the most prominent and important are *leaver_status, gold_per_min, leaver_status, xp_per_min, deaths, tower_damage* , etc. The full list can be found in the following chapter in which clustering is performed, i.e. postCluster.

3.2 First view and data pre-processing

As a data source, we used Kaggle, which is the largest platform for data science . Kaggle allows data scientists and developers to participate in machine learning competitions, write and share code with others, host databases, and much more [9]. The data set consists of dozens of columns and about 500,000 rows. Although the clustering process does not require much computing power, we tried to get the most descriptive and sound columns despite the large number of rows and columns.

We start by analyzing the data set from which we first remove the *match_id and account_id columns* because they have no weight in clustering, so they are 'unintelligent' data. We see that the *hero_id column* belongs to a corresponding hero, and each hero in the game is distinguished by its respective attributes, so we have heroes that cause damage, that heal, that push towers, and so on, these data are extracted from the first view of the data set.

We continue with the *player_slot column* which we do not take into account either, as it only serves to show the hero's inventory. The next nine columns are very descriptive and will all be part of our clean data set, those columns are *gold, gold_per_min, xp_per_min, kills, deaths, assists, denies, last_hits, hero_damage, hero_healing, tower_damage* and *level* . While the other columns describe the player's tools which are not descriptive because they do not change depending on the role. Anyway, now we have a data set of the type (Fig 6):

```
[{
  "gold": 1821.1461434370772,
  "gold_per_min": 447.6417456021651,
  "xp_per_min": 494.8359269282815,
  "kills": 9.657307171853857,
  "deaths": 9.190121786197563,
  "assists": 9.746278755074425,
  "denies": 5.939445196211096,
  "last_hits": 165.67016238159675,
  "hero_damage": 13487.634641407307,
  "hero_healing": 740.771650879567,
  "tower_damage": 1449.6248308525035,
  "level": 19.41982408660352
},...]
```

Fig 6: Sample data set

Let's now look at the data set schema, as we need to know the data types in order to change something or continue further ().

```
[ { "column": "gold",           "type": "DoubleType"    },
  { "column": "gold_per_min",   "type": "DoubleType"    },
  { "column": "xp_per_min",     "type": "DoubleType"    },
  { "column": "kills",          "type": "DoubleType"    },
  { "column": "deaths",         "type": "DoubleType"    },
  { "column": "assists",        "type": "DoubleType"    },
  { "column": "denies",         "type": "DoubleType"    },
  { "column": "last_hits",      "type": "DoubleType"    },
  { "column": "hero_damage",    "type": "DoubleType"    },
  { "column": "hero_healing",   "type": "DoubleType"    },
  { "column": "tower_damage",   "type": "DoubleType"    },
  { "column": "level",          "type": "DoubleType"    }]
```

Fig 7: Data set columns with corresponding types

We see that all data types are of type *Double* which means that different numerical metrics can be applied and this is what we need. We also see the statistics of our data set (Fig 8):

```
{
  "rows": "110",    "columns": "12"
}
```

Fig 8: Rows and columns of the pre-processed data set

So, our processed data set contains 110 rows and 12 columns. We only have 110 rows because there are 110 heroes, as we have grouped the data set based on heroes, while the 12 columns are the attributes we selected.

3.3 Implementing pre-processing in Scala

First we load the data downloaded from Kaggle via Spark using the commands (Fig 9):

```
var players = spark.read
    .option("header", true)
    .option("inferSchema", true)
    .csv(System.getenv("raw_kaggle_data"))
```

Fig 9: Loading the data set

So, *players* is currently a data set with 500,000 rows and about 30 columns, while *raw_kaggle_data* is the name of the environment variable that we have named for convenience. First, we select the specified columns and filter those whose level *is* greater than zero (Fig 10).

```
players = players
            .select(
              "hero_id","gold", "gold_per_min", "xp_per_min", "kills", "deaths",
              "assists", "denies","last_hits", "hero_damage", "hero_healing",
              "tower_damage", "level")
            .where(col("level") > 0)
```

Fig 10: Selecting and filtering in a data set

Then we apply them to the *players* dataset loaded above. Now since *players* is an immutable element , *we overwrite the players* dataset with the filtered dataset. It is a good idea to filter the dataset at a level greater than zero since such a level in the game does not exist and the space is filled as corrupted by the video game developers. The next step is to group and average each column, which is done as follows (Fig 11):

```
var groupedBy = players.groupBy("hero_id")
    .mean().drop("hero_id")
```

Fig 11: Grouping and removing the hero_id column

and then we just locally stored the data set in a convenient location for later use. The functions presented above, i.e. *mean* and *drop* are provided to the data set collection after using *group by* [10], this is because the average is the most worthy value to represent the respective column, in our case *mean* finds the arithmetic mean for each numeric column, this function is also added to the removal of *hero_id* because after grouping, we also create an *avg(hero_id)* , which is nothing more than a duplicate of the previous one.

So now we have the data set ready to apply other metrics or to apply the main algorithm, which is the purpose of this application.

3.4 Algorithm Application

We reload our clean dataset via the Spark library in order to apply clustering. Recall that this new dataset contains 110 rows and about 13 columns. First, we load the cleaned dataset through these rows (Fig 12).

```
var groupedBy = spark.read
    .option("header", true)
    .option("inferSchema", true)
    .csv(System.getenv("kaggle_data"))
```

Fig 12: Reading during algorithm application

and remove the inappropriate names from the respective columns. By inappropriate names we mean that if we apply metrics such as group *by*, or by average as we have in our case, then the function prefix is added to the columns, in our case, each column is named *avg(...)* .

And using the *RemoveBadNaming function* we created, the column will get the name it had before the function. We apply this metric to all columns. This function looks like this (Fig 13):

```
groupedBy = RenameBadNaming(groupedBy)
```

Fig 13: Function for renaming a data set

where *groupedBy* represents the filtered and now renamed data set. After this step, we construct the phases through which our algorithm will go. These phases look like the following (Fig 14):

```
        val bucketizer = new Bucketizer()
            .setInputCol("kills")
            .setOutputCol("kills_out")
            .setSplits(
              Array(Double.NegativeInfinity,
              3.0, 6.0, 9.0, 12.0,
              Double.PositiveInfinity))
        val imputer = new Imputer()
            .setInputCols(Array("hero_damage"))
            .setOutputCols(Array("hero_damage_out"))
            .setStrategy("median")
        val assembler = new VectorAssembler()
            .setInputCols(args)
            .setOutputCol("pre-features")
        val scaler = new StandardScaler()
            .setInputCol("pre-features")
            .setOutputCol("features")
            .setWithStd(true)
            .setWithMean(false)
        val kmeans = new BisectingKMeans()
            .setK(5)
            .setMaxIter(25)
            .setSeed(1)
        val pipeline = new Pipeline()
            .setStages(Array(bucketizer, imputer, assembler, scaler, kmeans))
```

Fig 14: The stages through which the algorithm has gone

From which we can see that we use six main classes, where:

- The first class, *Bucketizer* , enables grouping of data from the *kills column* into the *kills_out output column* . It is also seen that *setSplits is used as a method.* which allows the column to be grouped according to the partitions we have specified. This grouping helps in further clustering the data, since the *kills column* only has values from zero to an average of 50. Specifically, the partition belongs to the following groupings:

1. From the negative infinitive interval up to 3, the data is grouped at the value 0.
2. From the interval 3 to 6, the data is grouped at the value 1.
3. From the interval 6 to 9, the data is grouped at the value 2.
4. From the interval 9 to 12, the data is grouped at the value 3.
5. From the interval 12 to the positive infinitive, the data is grouped at the value 4.

- The second class called *Imputer* is used to fill in the blank values in the data set, where in our case the *median strategy is used* on the *hero_damage attribute* . In our specific case we do not find any application of this but it is used as a demonstration.

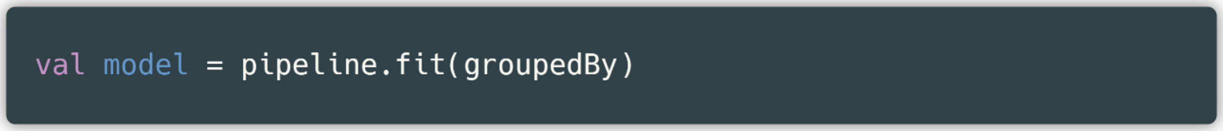
- The third class *VectorAssembler* that allows the collection of attributes into a single attribute called *features* in our case. This function is needed in every smart algorithm because the way they work is such that the algorithm requires an input column and provides an output column. The function therefore has two methods, *setInputCols* that requires a string with the elements we want to collect into an attribute and the *setOutputCol method* from which the output column is provided.

- The fourth class is *StandardScaler* and is used to scale the data. This type of scaler, among many of its kind, serves to convert each value in the range -1 to 1. As an input column we have taken *the pre-features* prepared from the previous step and as an output column we have *features* .

- The fifth class is called *Bisecting. Kmeans* and is the main function by which the clustering algorithm is applied. Among many other methods with default values, we are applying two new parameters to the *setK method* . This method sets the number of cluster center points which in our case is 5. In addition, we also have the *setMaxIter method* which sets the number of loops which in our case is 25, specifically this method tries 25 possible combinations of the algorithm and selects the best result.

- The sixth class is called *Pipeline* and it allows all the auxiliary classes to be grouped into stages and executed one after the other automatically before the algorithm is used. This technique makes it easier to use the main algorithms and makes the code more concise, readable and flexible. If we did not have this technique, we would have to apply each of the above classes to the data set and this is impractical and redundant.

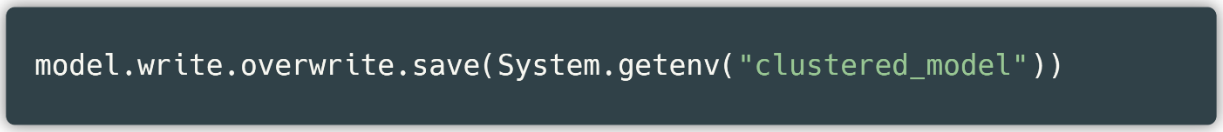
By applying these algorithms to our data set using the function (Fig 15):



```
val model = pipeline.fit(groupedBy)
```

Fig 15: Using phases (pipes) according to the fit function

All we have to do is save the model locally (Fig 16):



```
model.write.overwrite.save(System.getenv("clustered_model"))
```

Fig 16: Saving the model locally

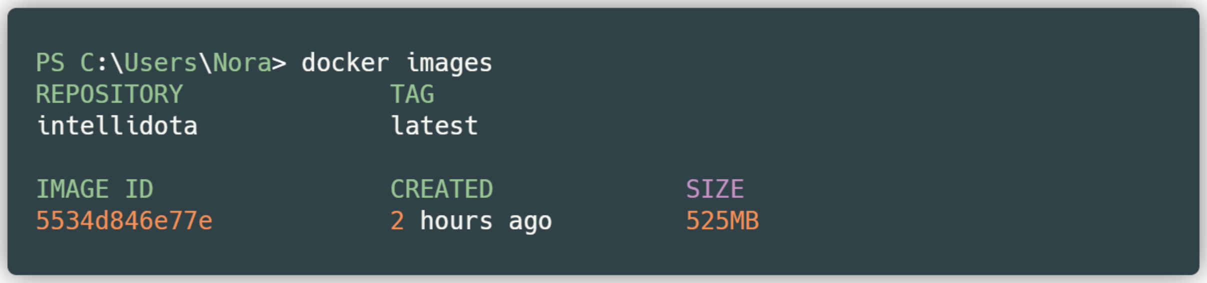
We store the model locally because it is more logical and practical to store the clustered model, so that if we want to extract statistics or apply functions to it, we do not have to train the model again as it requires time and computing power. This also allows us to perform clustering in real time, which means we can cluster any new row in a very short time.

3.5 Publishing Docker image to Google Cloud Platform

Before we continue, we need to describe what Docker is and how it works. Docker is a tool that helps in creating, developing and publishing web applications. It does this by packaging our software in packages called containers [11]. Containers are isolated from each other, which means that they have complete independence. The motivation for Docker, among many other reasons, came from the fact that to execute the program from two different systems it is much easier to create a Docker image that can be executed from the other side than to copy the source code. The other and main reason is that to publish a system on the web, which in our case was the publication of APIs created by the backend, Docker is the most suitable tool, especially when the application is distributed in several parts, where in this case we have the data sets, the trained models for classification and clustering, and others.

In addition to Docker, we also have Google Cloud Platform, which is a powerful asset offered by Google to perform many operations, ranging from machine learning, a repository for various files, DevOps tools like K8, and others.

What we have done with Google Cloud Platform is hosting the APIs provided by us. Why did we need something like this? For the reason that the application does not depend on the local execution of the application, so the frontend calls are performed regardless of whether the application is being executed locally, since the services of the application that are started by a local machine are lost if it is turned off. With all these details mentioned, we move on to the implementation part of this procedure. From the IntelliDota Classification part, once we have created a Docker image, we verify whether one exists (Fig 17):



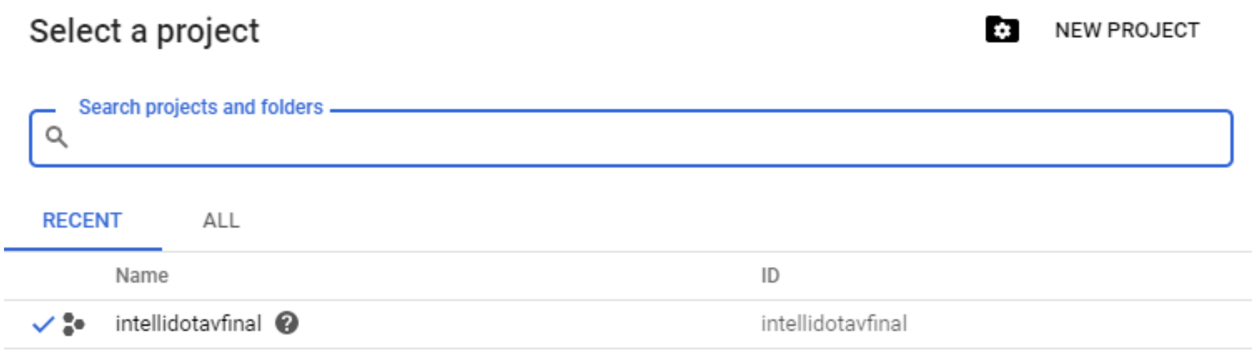
```
PS C:\Users\Nora> docker images
REPOSITORY             TAG
intellidota            latest

IMAGE ID               CREATED             SIZE
5534d846e77e           2 hours ago         525MB
```

Fig 17: Listing Docker images

So we see that we have an image called intellidota with an ID number of 5534d846e77e and a size of about 525 MB. That's all we need to move it to Google Cloud.

Let's move on to the Google Cloud Platform section [12], where the first step to take is to create a project, in our case, we have just created something like this named *intellidotavfinal* . Creating a project can easily be done through the main interface as follows, namely the *New Project button* . It can be seen that we have selected the *intellidotavfinal project* , so let's continue (Fig 18).



Select a project

NEW PROJECT

Search projects and folders

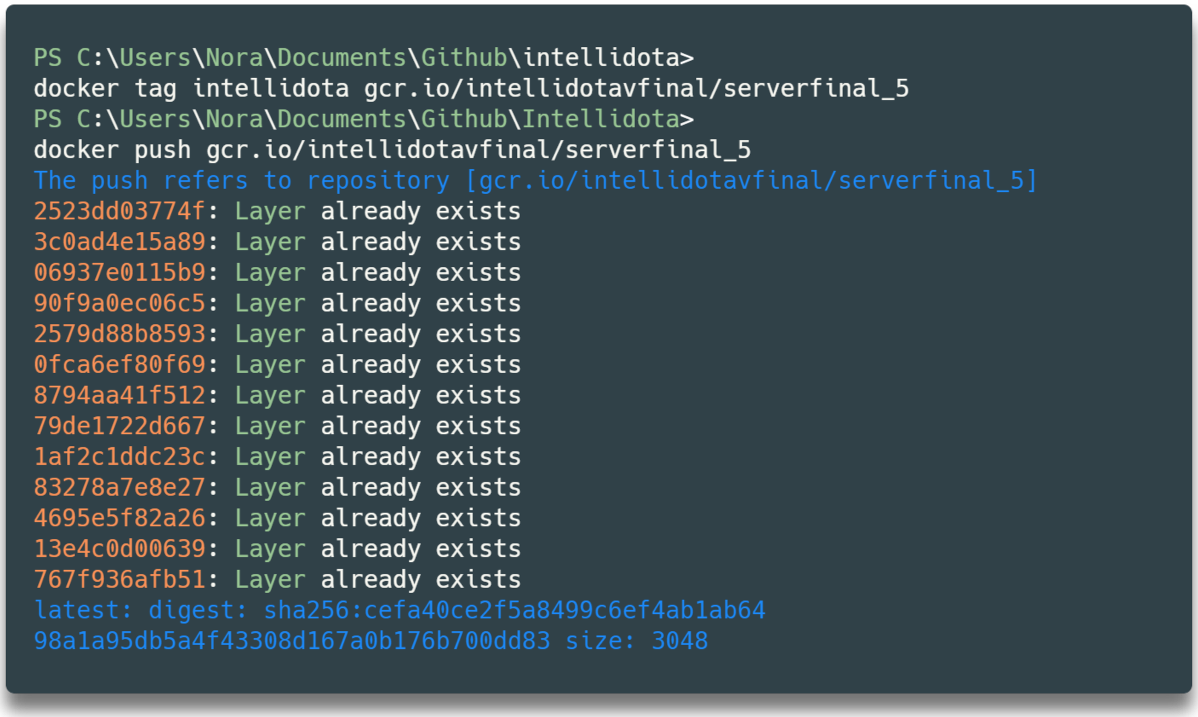
RECENT   ALL

	Name	ID
✓	intellidotavfinal	intellidotavfinal

Fig 18: Project selection and creation options

Once the project is created, we open the navigator and select the Cloud Run option. Essentially, Cloud Run is a platform that allows manipulation with containers, in our case with Docker images.

Then, using a CLI (*Command Line Interface*), we publish our local image to Google Cloud Platform (Fig 19).



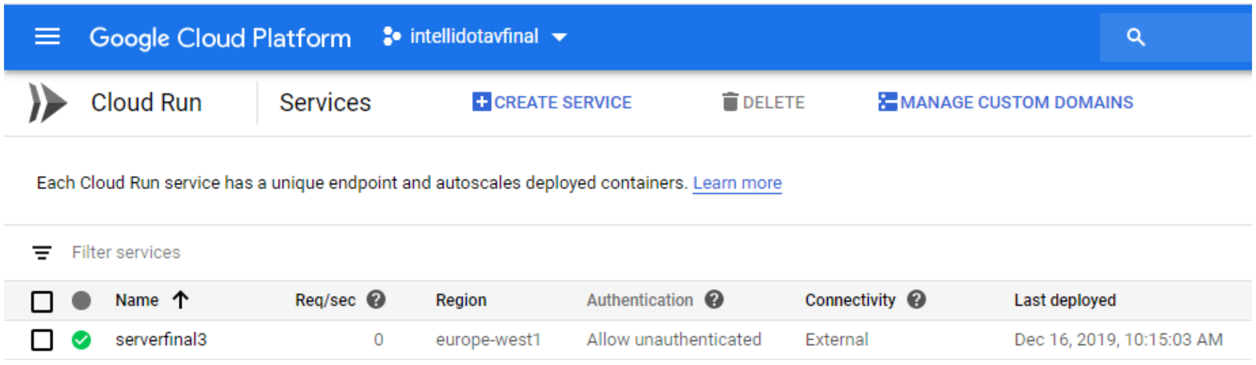
```
PS C:\Users\Nora\Documents\Github\intellidota>
docker tag intellidota gcr.io/intellidotavfinal/serverfinal_5
PS C:\Users\Nora\Documents\Github\Intellidota>
docker push gcr.io/intellidotavfinal/serverfinal_5
The push refers to repository [gcr.io/intellidotavfinal/serverfinal_5]
2523dd03774f: Layer already exists
3c0ad4e15a89: Layer already exists
06937e0115b9: Layer already exists
90f9a0ec06c5: Layer already exists
2579d88b8593: Layer already exists
0fca6ef80f69: Layer already exists
8794aa41f512: Layer already exists
79de1722d667: Layer already exists
1af2c1ddc23c: Layer already exists
83278a7e8e27: Layer already exists
4695e5f82a26: Layer already exists
13e4c0d00639: Layer already exists
767f936afb51: Layer already exists
latest: digest: sha256:cefa40ce2f5a8499c6ef4ab1ab64
98a1a95db5a4f43308d167a0b176b700dd83 size: 3048
```

Fig 19: Tag association and push to Cloud

The first command associates our project with the gcr.io link, where GCR stands for Google Cloud Repository, and then we push our project through the associated link. The following link consists of the GCP project name, which in our case is *intellidotavfinal* , and *serverfinal_5* is the name of the branch, or super version, that we are publishing to.

This process, i.e. uploading, takes approximately 10 minutes as the Docker image size is around 500 MB. Below we notice that some messages appear like *Layer already exists* , this happens because we have just published the same version earlier (the version in which the server is currently running).

After performing this action, we switch back to GCP and continue the process in Cloud Run. First, we create a new service via the *Create Service button* (Fig 20).

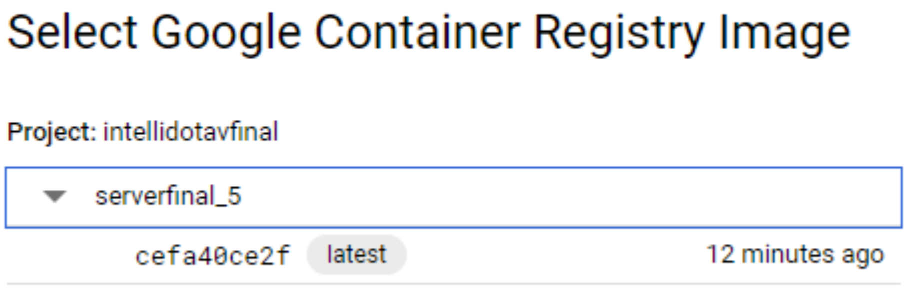


Each Cloud Run service has a unique endpoint and autoscales deployed containers. Learn more

		Name	Req/sec	Region	Authentication	Connectivity	Last deployed
☐	✓	serverfinal3	0	europe-west1	Allow unauthenticated	External	Dec 16, 2019, 10:15:03 AM

Fig 20: Possibility to create a Cloud service

By clicking this button, a window appears in which we must select the container from the list of all containers published by us (Fig 21, we select the container published earlier, i.e. the super version serverfinal_5 with the *latest tag* – automatically associated with Google).


Select Google Container Registry Image

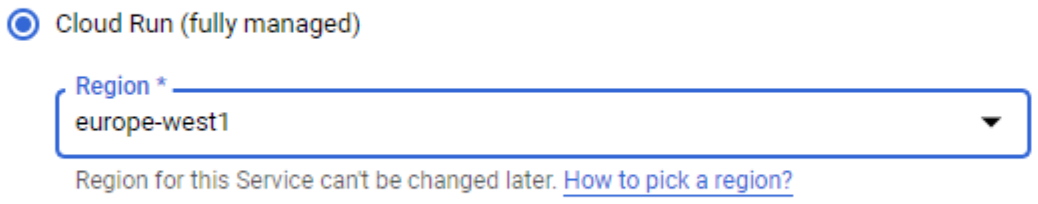
Project: intellidotavfinal

serverfinal_5

cefa40ce2f latest 12 minutes ago

Fig 21: Image published by CLI

We set the region so that calls are made faster, by this we mean that if we set a region that is far away from where we are connected to the internet, the time for transmitting and receiving data through the network is high (Fig 22).

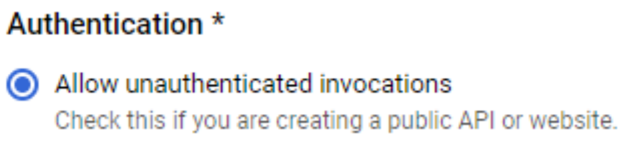

Cloud Run (fully managed)

Region *
europe-west1

Region for this Service can't be changed later. How to pick a region?

Fig 22: Selecting the closest region

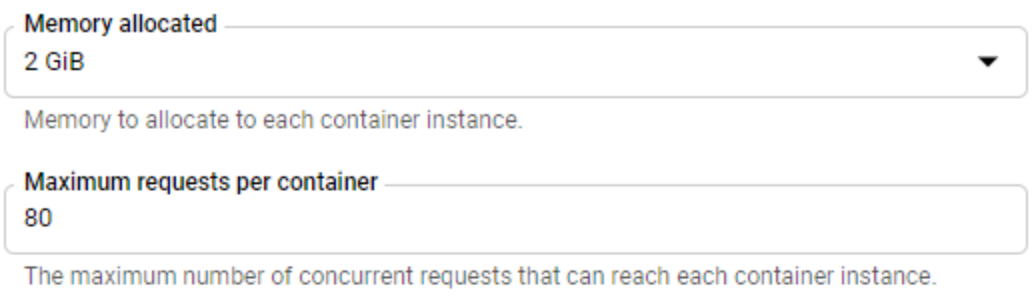
We allow calls to our services to be made even if they are not authenticated. This way, anyone who wants to can access the Json documents directly using the actual connection we have provided (Fig 23).


Authentication *

Allow unauthenticated invocations
Check this if you are creating a public API or website.

Fig 23: Allowing unauthorized access

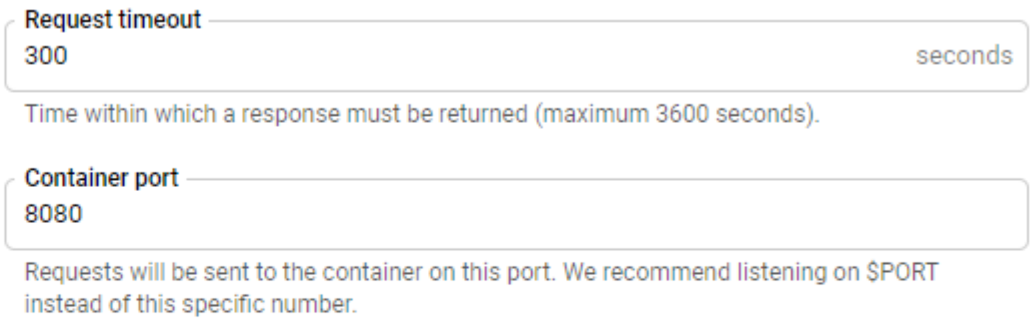
We allocate 2GB of memory to our container because that is how much Spark requires in the backend, which means that if it is allocated less memory, processing cannot occur, and errors will be thrown, since more processing power is needed. In addition, we also allow the maximum number of simultaneous requests to be 80 (Fig 24). This practice is applied in order to avoid various attacks, among the most famous of which is denial of service (*DOS – Denial Of Service Attacks*).


Memory allocated
2 GiB
Memory to allocate to each container instance.

Maximum requests per container
80
The maximum number of concurrent requests that can reach each container instance.

Fig 24: Allocated Cloud Memory and Maximum Requests

Next , we limit the time for the responses to be returned, which in our case is 300 seconds, and specify the container port, which Google has assigned to 8080 (Fig 25). This can be changed, but it should be noted that this port and the port on which the Docker image is built must be the same, to always achieve this dynamically, we have specified the port in the Dockerfile as described in IntelliDota Classification as an environment variable.


Request timeout
300 seconds
Time within which a response must be returned (maximum 3600 seconds).

Container port
8080
Requests will be sent to the container on this port. We recommend listening on $PORT instead of this specific number.

Fig 25: Host port in Cloud

So if this process is performed correctly, our service will be accessible on port 8080. With this information, we click the Create button and wait for our service to be created.

If we made a mistake during the procedure, this image appears along with the error during the procedure (Fig 26).

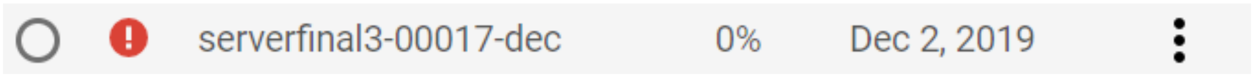

serverfinal3-00017-dec 0% Dec 2, 2019

Fig 26: Image execution with errors during the procedure

Otherwise , we say that our services are available on the website specified by GPC, exactly as the following image appears (Fig 27).

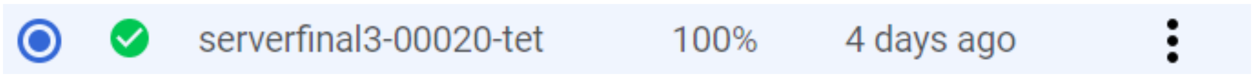
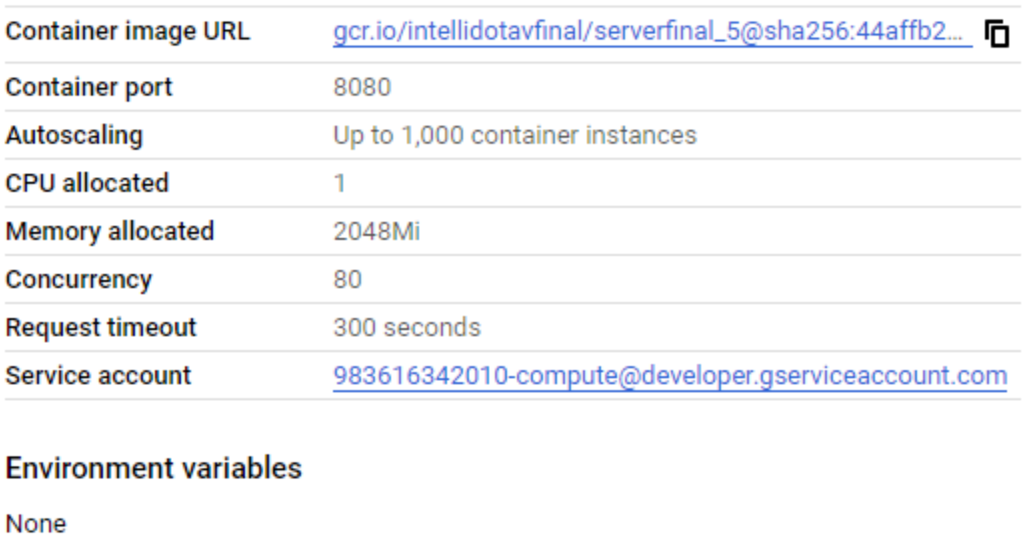

serverfinal3-00020-tet 100% 4 days ago

Fig 27: Image execution without errors during the procedure

If we are interested in seeing more data, we have the right side of the screen, so (Fig 28):



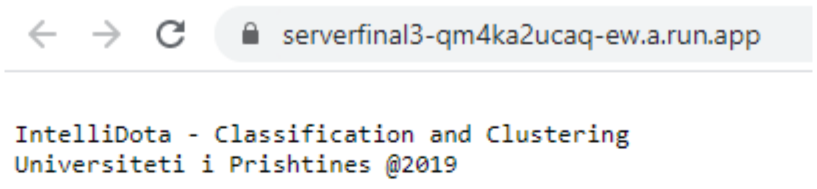
Container image URL	gcr.io/intellidotavfinal/serverfinal_5@sha256:44affb2...
Container port	8080
Autoscaling	Up to 1,000 container instances
CPU allocated	1
Memory allocated	2048Mi
Concurrency	80
Request timeout	300 seconds
Service account	983616342010-compute@developer.gserviceaccount.com

Environment variables

None

Fig 28: More details about hosting

Finally we have the link https://serverfinal3-qm4ka2ucaq-ewë.a.run.app which if we try to access, we see (Fig 29):



```
IntelliDota - Classification and Clustering
Universiteti i Prishtines @2019
```

Fig 29: Final connection accessible at any time
So, our services are accessible by anyone at any time.

4 Application Development

The application is developed in the Dart programming language, [13]which is an object-oriented language, using the Flutter SDK. Flutter is an open-source mobile application development framework created by Google. It is used to develop applications for Android and IOS, and is the primary way applications are created for Google Fuchsia. Flutter, with its creation method, makes the work extremely easy by giving developers numerous options, which creates a good experience and also speeds up development time. It supports several environments for building applications, which are IntelliJ, Visual Studio Code and Android Studio. One of the features of Flutter is the reloading of the application, which allows us to observe any changes made to the code in real time.

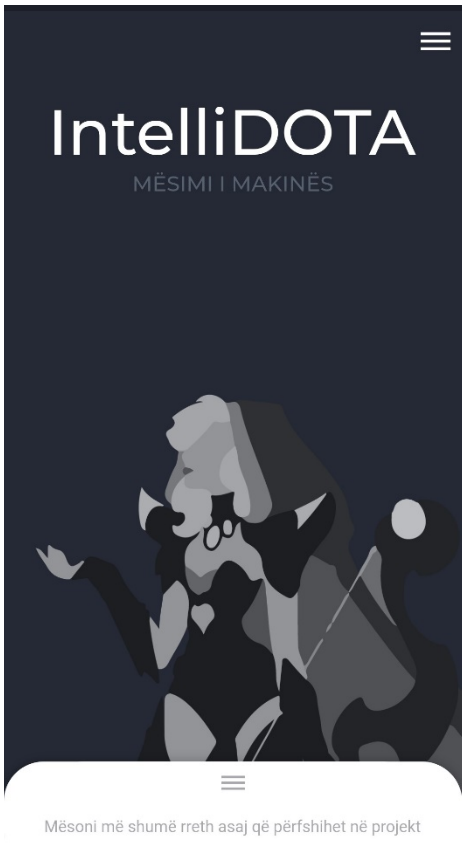


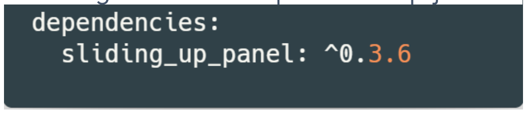
Fig 30: Pamja hyrëse

Based on the endpoints created in Scala, special calls were created in Flutter using the *http package* developed by Dart, enabling the creation of data visualizations.

To best demonstrate the operation and structure of Flutter, we start with the application's introductory view, which provides general details about its main function (Fig 30). When the application is opened, the user is first presented with the application's home page, which is built using components (Widgets *)* . which are the main features on which everything in Flutter is built. In this section we can get a brief understanding of what is included in the project, this is achieved by dragging the bottom panel (Fig 31), which is created using the following package which provides a ready-made *Widget that provides this view.*

To navigate to other parts of the application, we can use the menu located at the top of the home page, which will open the view with the list of screens, where we have a total of eight possible screens to which we can navigate.

Fig 31: Vartësia për rrëshqitje



```
dependencies:
  sliding_up_panel: ^0.3.6
```

4.1 Menu options

4.1.1 The possibility of choice

Since we have two trained datasets, the metrics can be applied to the dataset we created ourselves, i.e. Steam or Kaggle. For which dataset we want to operate on, we have created two buttons (Tabs , Fig Fig 32through which the type of the respective dataset is set. It should be noted that this option does not exist in the Clusters menu and in the Cluster Grouping menu, which only function for clustering. As an example, we have chosen 'Clustering', which means that the metrics are operated on the Kaggle dataset and as columns are automatically aligned to the respective dataset, in our case the Kaggle dataset.

Fig 32: Pamja e butoneve për klasifikim / kllasterim

4.1.2 Discretizer

The discretizer is one of the most useful tools in our toolbox, representing the transformation of continuous values into countable values. Since the datasets consist of continuous values of high magnitude, it was reasonable to build such a tool that discretizes the data into one of the groups selected by the *slider* . This screen can be opened by selecting *Discretizer* from the menu section as shown below (Fig 33):

Fig 33: Diskretizuesi

On this screen we have the opportunity to see the data distribution graphically.

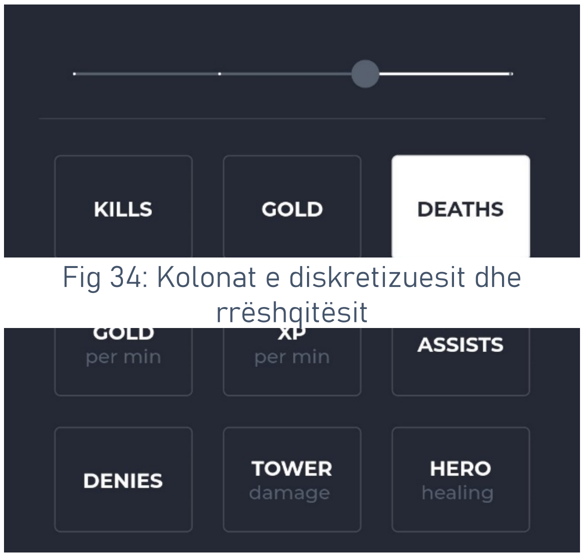


Fig 34: Kolonat e diskretizuesit dhe rrëshqitësit

First, from the list displayed at the bottom of the screen, we select one of the columns for which we want to make this visualization and then, using the slider , we can decide on the number of divisions in which the values of the respective column will be grouped (Fig 34). We have decided to allow the possibility of choosing one of the 4 different values of the slider, since from the assessments made, these values best represent the voluminousness of the data.

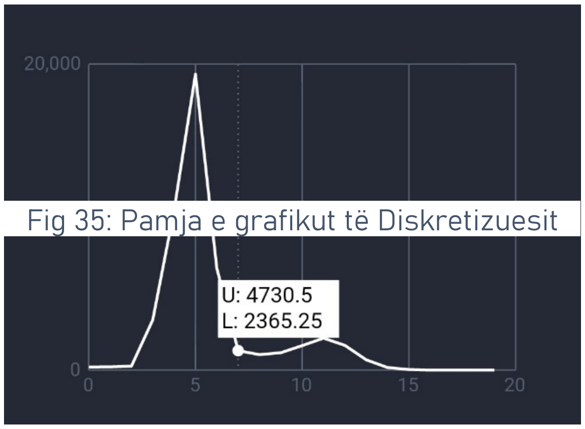
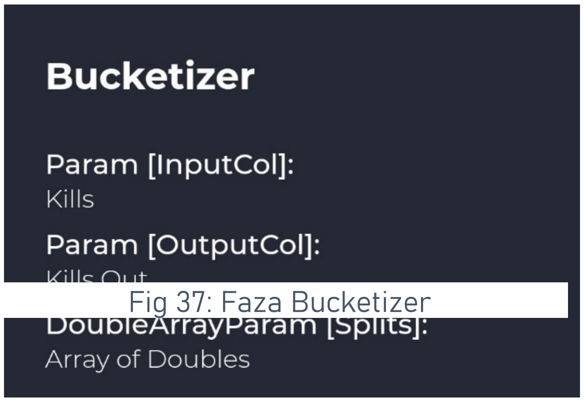


Fig 35: Pamja e grafikut të Diskretizuesit

By selecting one of the columns, the request will be processed and as a result a graphical representation will appear which demonstrates on the X axis the number of clusters selected by us, be it 5, 10, 15 or 20 clusters and on the Y axis the amount of clustering (Fig 35). To see what is the range of the grouped values, we click on any point on the line and the lowest and highest value appears, where the letter U stands for Upper *and* L for Lower . As an example we have the image in the fig where the lowest value is 2365.5 and the highest value 4730.5, where in our case there are about 2000 values as observed on the Y axis, while the clustering belongs to the bucket with number 7 on the X axis. This value was obtained from the data collected for classification, using the *gold_per_min attribute* . It should be noted that this screen is built using the *getGroupAndCount API* .

4.1.3 Training Phases

During data pre-processing to algorithm selection, in order to accurately see the stages our algorithms went through until they were trained, we visualized them in order from top to bottom, meaning that the highest stage was executed first while the stage below represents the last stage.



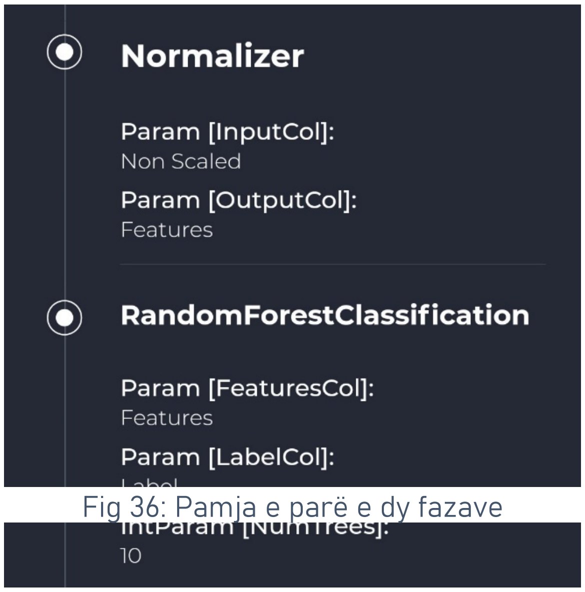
Bucketizer

Param [InputCol]:
Kills

Param [OutputCol]:
Kills Out

DoubleArrayParam [Splits]:
Array of Doubles

Fig 37: Faza Bucketizer



Normalizer

Param [InputCol]:
Non Scaled

Param [OutputCol]:
Features

RandomForestClassification

Param [FeaturesCol]:
Features

Param [LabelCol]:
Label

IntParam [NumTrees]:
10

Fig 36: Pamja e parë e dy fazave

If, for instance, we want to examine what methods are used in creating a trained model, we have, for example, a Bucketizer where it requires several parameters such as input columns, output columns and the number of partitions (Fig 37). It should be noted that if a method is used two or more times, a random number is added to the method used later, for example, if we use another method of Bucketizer then the other method is named Bucketizer4 (or a random number).

A set of these methods forms a whole pipeline *that* produces a trained model. If we consider a pipeline, for example Classification, we can say that first the data is normalized, where as an input parameter we have a number of columns and as an output, it gives a single column called Features where the values are scaled between 0 and 1. After that, the Random Forest Classifier algorithm is used with the number of trees 10 (Fig 36). Continuing in this way, the whole pipeline is constructed.

Funnels are useful not only for optimizing the process but also for readability. The funnel is dynamically formed and depending on the methods we use during the initial machine learning period, the corresponding stages are created. This screen is made using the *getStages API.*

4.1.4 Data Samples

To better display the collected data to the user, we have created a list with the names of the attributes of the respective data sets, a list that is dynamically generated, where each element includes a percentage (20%) of the values that provide a representation of the type of data that extends across the fields (Fig 39, Fig 38). An example of the generated and visualized columns looks like this.

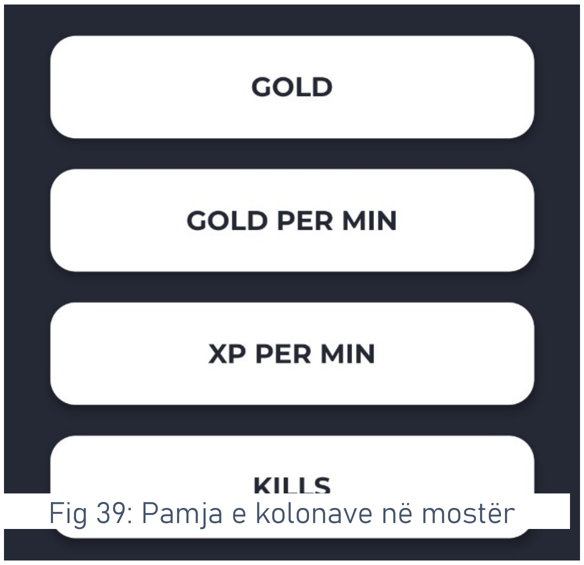
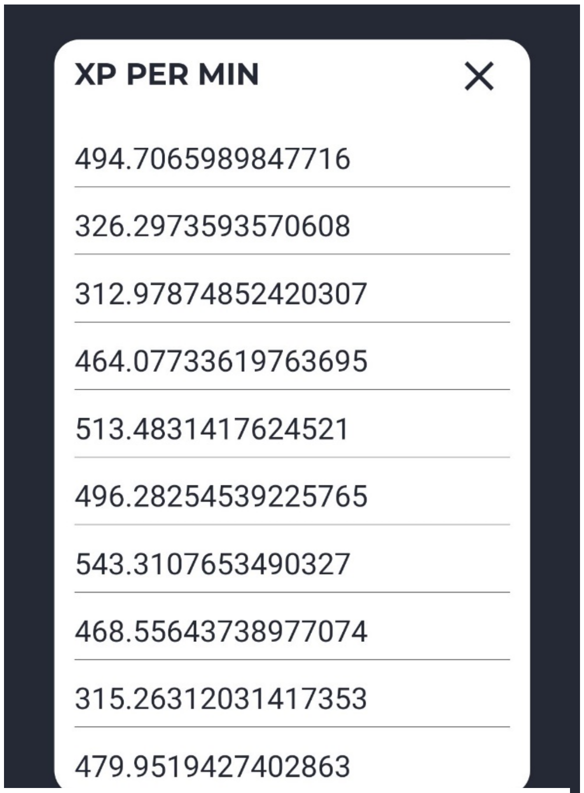


Fig 39: Pamja e kolonave në mostër



494.7065989847716
326.2973593570608
312.97874852420307
464.07733619763695
513.4831417624521
496.28254539225765
543.3107653490327
468.55643738977074
315.26312031417353
479.9519427402863

Fig 38: Pamja e vlerave në mostër

While clicking on the *XP PER MIN attribute* will open a field with the values of this attribute, the first impression of which suggests that they are decimal numerical values and that they belong to a range from 300 to 600. This screen is realized using the *getSample API* which accepts two parameters, the data set which in this case is Kaggle and a percentage which in this case is 20. Since it is difficult to visualize the data of a sample in a table on an iOS or Android operating system, then due to the appearance, it was considered more reasonable to display the columns vertically.

4.1.5 Correlation Matrix

The correlation matrix is a type of table that shows the correlation coefficients between variables [14]. Each cell in the table shows the relationship between two variables. The matrix is square in shape, showing the same attribute names on both axes. It serves to summarize large amounts of data, where the main goal is to obtain a model, and where in our case the Clustering data set it is observed that the attributes are highly correlated with each other, while in Classification the independence between the attributes prevails more. The coding of the correlations is of the range of values from -1 to 1, where 0 (Fig 40) indicates that the values do not correlate with each other at all, 1 means that the variables correlate highly while -1 means that there is a high inverse correlation.

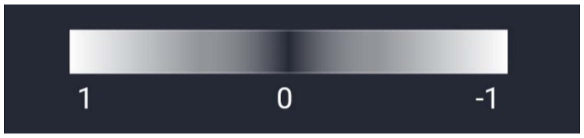


Fig 40: Vlerat fundore të korrelacionit

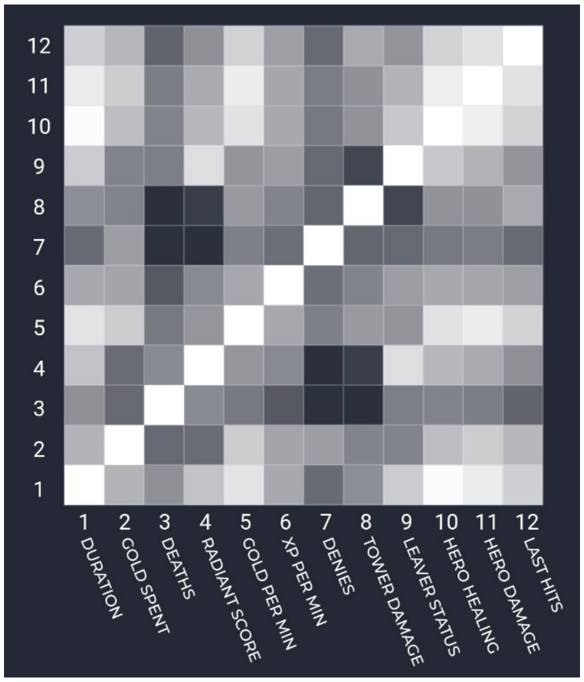


Fig 42: Matrica e korrelacionit të data setit Kaggle

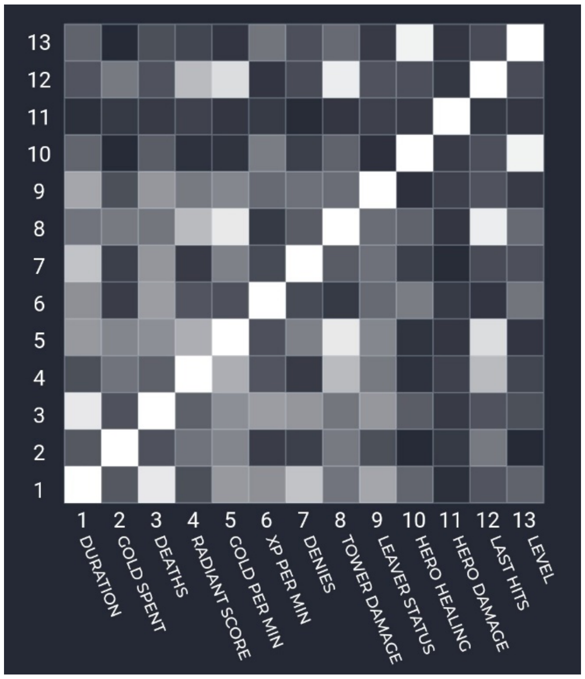


Fig 41: Matrica e korrelacionit të data setit Steam

Coding can also be done in other ranges, but this does not have a major effect since any value could adequately reflect the dependency.

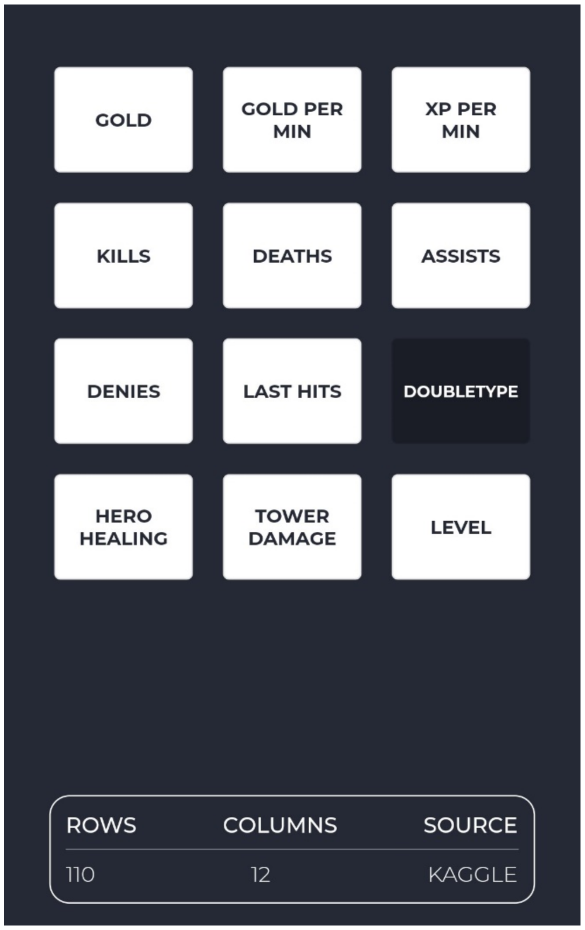
To scale the variables, no library was used for this purpose, but the *Correlation method* in Scala has a unique scaler built into it. Since Flutter did not have any ready-made *Widget* for creating the correlation matrix from the package used for visualization, an advanced Flutter class, *CustomPaint* , was used, which provides a canvas *for* drawing different shapes that could not be created correctly using the existing *Widgets . So from this class we inherit and use the paint method, to which we also write the functions to dynamically draw the dependency cells. This screen is implemented using the getCorrelationMatrix* API that accepts as a parameter the data set that we want to apply the matrix to (Fig 42, Fig 41).

4.1.6 Data Set Structure

In this part of the application we have provided a view that informs us about the number of attributes and fields used for each data set, as well as the ability to see the data type for each attribute, although almost all of them have the same data type and differ only from data set to data set. In the current case (Fig 43), we are using Kaggle as the data set from which we see that the type of the clicked attribute is Double while the data set consists of 12 columns and 110 rows. We recall that this data set is pre-processed and that the original data set contains approximately 500,000 rows and about 70 columns.

This screen is created by combining two APIs used, which are *getSchema* which allows listing columns along with their type, while *getStats* provides the number of columns, rows, and the source of the data set for which statistics were generated.

It should be noted that this screen requires the highest processing and opening time, as the calculation of rows and columns is done in real time, so with the click of a button, this data is recognized and displayed on the screen, which means that there are no values previously stored in the application.

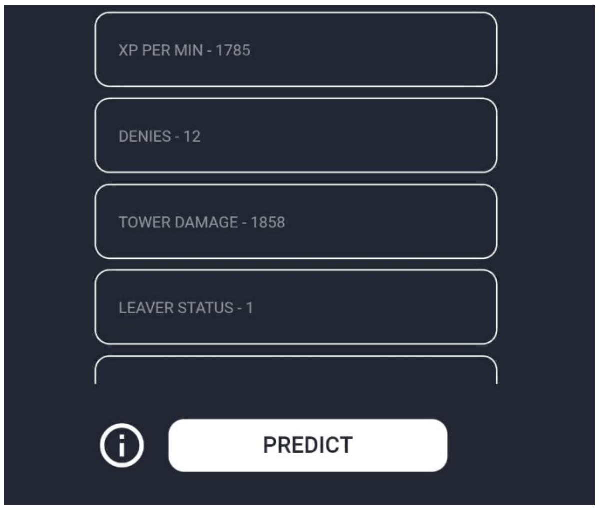


ROWS	COLUMNS	SOURCE
110	12	KAGGLE

Fig 43: Struktura e data seteve

4.1.7 Real-Time Classification and Clustering

Among the main parts of the application is real-time classification and clustering. By real-time we mean the ability to use trained models whenever we click the *Predict button* with the assigned columns. First view It looks like this for both classification and clustering. We see () that for each space in which we can write we have the name of the column that belongs to the space and a numerical value, which represents the previously defined value of the column. This screen consists of 3 final calls, which are:


XP PER MIN - 1785

DENIES - 12

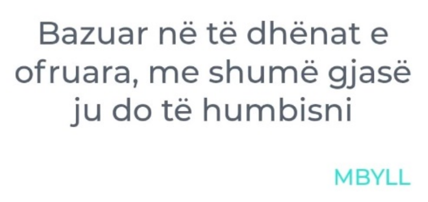
TOWER DAMAGE - 1858

LEAVER STATUS - 1

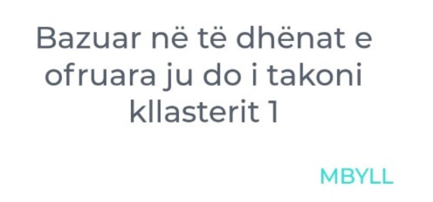
Fig 44: Pamja e klasifikimit në kohë reale

- postPredict - to perform classification where the columns of the data set are given as input and a text is provided as output indicating whether we won the game or lost.
- postCluster - to perform clustering where the columns of the data set are given as input and the cluster number in which the data is grouped is provided as output.
- getColumns – which lists the column names along with the first defined values.

Upon completion of the classification or clustering process, a text appears indicating what the predicted value is or which cluster the new values belong to.


Bazuar në të dhënat e ofruara, me shumë gjasë ju do të humbisni

At first glance, it can be seen that with these first defined attributes we will lose the game, as can be understood from the text that displays 'Based on the data provided, you will most likely lose'.


Bazuar në të dhënat e ofruara ju do i takoni klasterit 1

While with the first defined clustering attributes, i.e. the second view, we see that we will belong to cluster number 1, i.e. 'Based on the data provided, you will belong to cluster 1'.

What data cluster 1 has can be understood by reading the Clusters.

4.1.8 Clusters

The clustering screen graphically displays the clustered data in a certain order. We explain this menu based on the image below (Fig 45). This menu is valid only for the clustering process, so there is no scrolling option. We see that on the left, we have a series of names that represent the attributes of the Kaggle data set. Each attribute is associated with 5 different colors, where the order from top to bottom represents the cluster number, which means that the blue color belongs to cluster 1, the red color belongs to cluster 2, the yellow color to cluster 3, the green color to cluster 4 and finally the white color belongs to cluster 5.

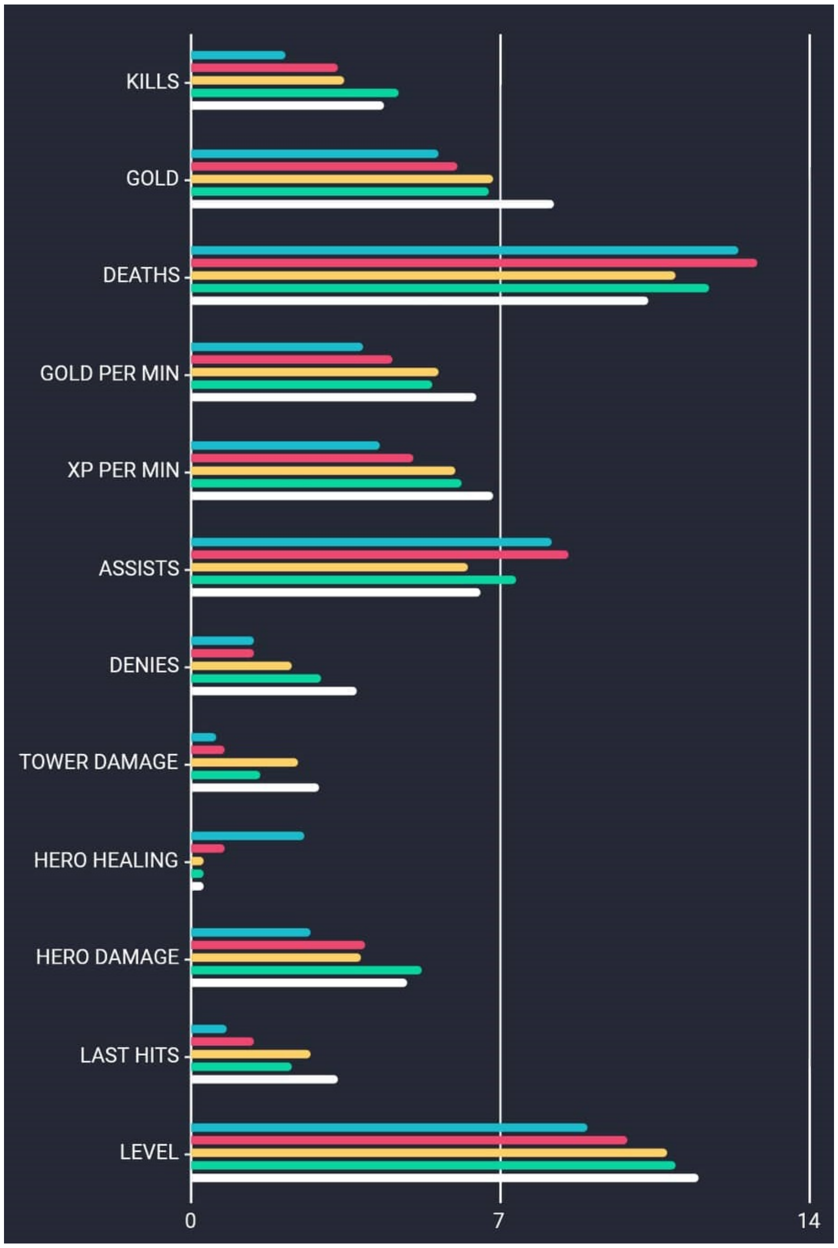


Fig 45: Kllasterët në mënyrë grafike

Once the clusters are identified, we begin to analyze them until we extract sound and practical data for each grouping. Recall that Dota 2 is a video game where players are divided into certain roles, something that we will also stimulate to extract depending on attributes.

We're starting with cluster 1, moving on, creating a table for each member.

4.1.8.1 Blue cluster - *Supporters curative*

From the graph, we see that this group is distinguished by very few kills , little money *,* many deaths *,* little money per minute *,* and many assists *.* With these statistics, we conclude that this group is not one of the killers, since the killers would have money with which to supply themselves with tools and kill other heroes, this means that with these abilities, the number of deaths would be smaller (since the protection itself would be greater). With a large number of assists, we say that this group belongs more to the supporters, but let's continue with further analysis to correctly name this group. We say that this group has few denials, causes little damage to towers, and heals excellently. With these data, we say that this group accurately belongs to the supporters without continuing further, since comparing this group with others in terms of healing has a clear advantage. A few last-ditch strikes and a low level only reinforce what we said. We conclude that this group belongs to the supporters, those who sacrifice themselves for another role.

We will call this group *the Supporters. healers* as they are clearly characterized by a different attribute. It should be noted that supports may not be healers all the time, in the game you can support a hero even without healing him.

4.1.8.2 Red cluster - *Second-hand supporters*

With few kills, few money and a considerably high number of deaths, we say that this group is not a killer either. However, based on the attributes, this group can still belong to a group that pushes towers, supports or is the first in line. We continue with a small number of experience per minute and a large number of assists. Few denials and few tower pushes. The possibilities that the group belongs to tower pushes are excluded. It remains to be seen whether it is a group that is the first in line or a support but not a healer. We continue with the attributes and we see that this group heals to a certain amount and causes damage to other heroes to a certain extent. It has few final blows and few levels. With these data, we say that this group also belongs to the supports, but in this case not healers, but *second-hand Supports* .

4.1.8.3 Yellow cluster – *First in line*

This group is characterized by average kills, and above average money, a low number of deaths and relatively high experience. With very few assists, many denials and high tower push, but little healing. With this information, the theory falls that this group could belong to some other group of supports, since they were distinguished for healing or assists and relatively little money and experience. There are still open possibilities that this group belongs to killers, tower pushers or first in line. We continue with the other attributes; Little damage to other heroes and many final blows, and high level. With this said, the name of this group could be *First in Line.* Based on the experience in the game, these heroes, in addition to these attributes, are distinguished for lone play (few assists) and push towers but not the most, although their role requires this.

4.1.8.4 Green Cluster – *The Killers / Playmakers*

A large number of kills, a large number of money and deaths are the characteristics that distinguish this grouping to continue with high experience and a large number of assists. A high experience and a large number of assists are not something that characterizes the supports, so this option is removed from the game. Many denials and considerable damage to towers but no healing. We continue with massive damage to heroes and a high number of final blows and a very high level. We can identify this grouping with the name *Killer* or *Playmaker* for several reasons. Killer since it has the attributes that identify this name, along with money and experience. While *Playmaker* since it has a large number of deaths and is not known for massive tower pushes. The number of assists helps this name that we have given it and also the high level.

4.1.8.5 White Clustering – *Goal Followers*

With a high number of kills, a maximum number of money and a small number of deaths, we see that this group of data is unique, since for each of the above attributes there is an ideal margin. With a lot of experience and an average number of assists, we say that this group definitely does not belong to the supports. We continue with a high number of denials and maximum number of tower damage, very little healing and a lot of damage to heroes. Many finishing blows and maximum levels. There are many designations that this attribute can have, but it cannot be grouped with the first in line since it has a high number of kills and damage to heroes. It can be included in the category of killers but even this designation does not identify this category ideally since it has maximum push to towers. This cluster has ideal data for each attribute, data that is only obtained if the game is played perfectly and objectives are followed correctly, so we will call it *Objective Chasers.*

If we summarize these groups and names, we say that:

Cluster	*Color*	*Name*
The	Blue	*Supporters curative*
of secondment	Red	*Second-hand supporters*
III	Yellow	*First in line*
IV	Green	*assassins / Playmakers*
V	White	*Goal trackers*

4.1.9 Cluster Grouping

Finally, in order to see some basic statistics for the clustering performed above, we have the Clustering screen (Fig 46). This screen is powered by the *getClusterCount API* which in principle represents a simple counter. From the figure we see that in:

- Cluster number 0, called *Supporters medical* , around 25 values (in light green) are grouped.

- Cluster number 1, called *Second-hand supporters* , is grouped around 32 values (in red).

- Cluster number 2, called *First in Line* , is grouped around 16 values (colored yellow).

- Cluster number 3, called *the Killers / Playmakers* , grouped around 24 values (in green)

- Cluster number 4, called *Goal Followers* , is grouped around 13 values (in white)

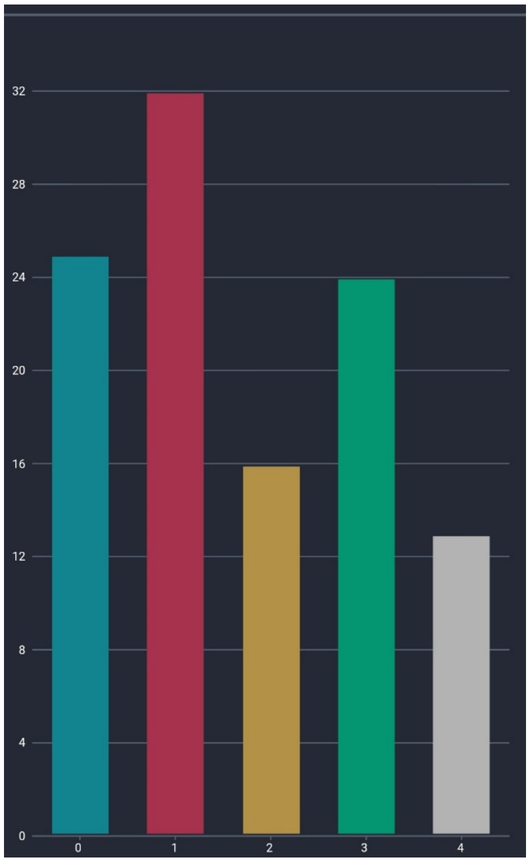


Fig 46: Grupimi i kllasterëve

5 Conclusion

The project is divided into two parts, the *frontend* and the *backend* , which is where the most difficulties came from, as the visualization and description of the main functions had to be done in full coordination. However, the experience of creating such a platform in which the *frontend* and *backend are coordinated* has been special and will come into great use in the future, as nowadays it is rare to work with *a frontend* integrated into the corresponding *framework* .

Although similar projects exist in the industry, none are like this. Among the most prominent projects that we have been inspired by are Opendota and Dotabuff, which integrate graphics with real-time statistics and provide a nice interface in which improvisation can be easily achieved, however, neither integrates machine learning into them, be it clustering or classification, or any other type of learning.

So, in IntelliDota Clustering, the Bisecting K-Means algorithm is used, which is a version of K-Means. This algorithm was used in our project with a K value of 5 and 25 iterations, so that the result is more accurate. We have saved the trained algorithm using a Docker image. We have applied this algorithm to a pre-processed data set, since the non-pre-processed data set contains 500,000 rows and 12 columns, and clustering on such a data set would not make sense.

Pre-processing mainly consists of grouping the data set on the hero_id column, a column that uniquely identifies the hero. So the normal data set consists of 500,000 rows, and each row describes the behavior of a hero for a given game. With such grouping, the size is reduced to 110 rows and as aggregation, we used the average. So the final data set on which the algorithm is applied contains 110 rows and 12 columns. From the first part of the IntelliDota project, it is known that there is an unpublished image in which the application programming interface is served, for both data sets.

Publishing this image combines Docker and Google Cloud SDK, so we upload the created image to Google Cloud Platform using Google SDK, which is a set of tools that enable uploading via a CLI. Publishing to the Cloud is done by assigning a nickname to the image and then pushing it to the server. This is followed by configuring it in the Google interface as described above.

The implementation of the application, carried out using Flutter, which uses the Dart programming language, includes the possibility of visualization, applying metrics, and analysis. Among all these options, more specifically, we have the possibility of discretization of data, where data with continuous values are grouped depending on the number we choose, be it 5, 10, 15, or 20, and a graph with the corresponding values according to the groups is displayed, we have the possibility of seeing the phases of the trained model, the possibility of seeing the correlation matrix, the possibility of prediction and clustering in real time, the possibility of seeing the cluster groups after scaling the values, and others.

Meanwhile, our project cannot currently be expanded further but can be used as a basic application for interesting statistics. However, we will continue to work with machine learning and extract deeper analyses both for video games and for concrete cases. In video games, a further step can be taken, such as extracting analyses of what were the main reasons that a team won. The level of machine learning that needs to be developed for this application is not more advanced than the level we are currently using, but the analytical capabilities need to be increased, since everything is again extracted from graphs.

Other important elements for both the project and the future have been the use of the Json structure, i.e. manipulating attributes via Scala, the use of Gson as an intermediary library between Json and Scala, and retrieving data via Scala dependencies, such as *lihaoyi ,* through which we have implemented the requests. It is worth mentioning once again the *ngrok application* through which we were able to publish our endpoints so that the second part, the *frontend , can be implemented* .

All parts of this application are public, starting from the created dataset, to Scala, the connection between them with the REST model, and the classification and clustering algorithm so that developers can be helped in any way they can.

6 Annexes

The source code and data set are listed below:

- Backend: https://github.com/LabinotVila/IntelliDota
- Trained models: https://github.com/noraibrahimi/IntelliDotaIC
- Frontend: https://github.com/noraibrahimi/IntelliDota-mobile
- Steam dataset: https://www.kaggle.com/labinotvila/dota-2-steam-api-fetched-dataset

7 List of possible end calls

7.1 index

host / index

Parameters:	
Description:	Returns the content of the start page, which is the list of endpoints provided by us.

Tab 1: Endpoint index

7.2 getColumns

host / getColumns

Parameters:	kind	– dataset type [steam / Kaggle]
Description:	returns a list of columns of the corresponding dataset	
Example:	/getColumns? kind = steam	

Tab 2: getColumns endpoint

7.3 getSample

host / getSample

Parameters:	kind	– dataset type [steam / Kaggle]
	percentage	– percentage of the monster [0 to 100]
Description:	returns a sample of the corresponding dataset, depending on the percentage	
Example:	/getSample? kind =steam& percentage =10	

Tab 3: getSample endpoint

7.4 getStages

host / getStages

Parameters:	kind	– dataset type [steam / Kaggle]
Description:	returns an array of the stages the data has gone through	
Example:	/getStages? kind =kaggle	

Tab 4: getStages endpoint

7.5 getCorrelationMatrix

host / getCorrelationMatrix

Parameters:	kind	– dataset type [steam / Kaggle]
Description:	returns a string of numbers indicating the relationship between columns of the corresponding dataset	
Example:	/getCorrelationMatrix? kind =kaggle	

Tab 5: Endpoint: getCorrelationMatrix

7.6 getGroupAndCount

host / getGroupAndCount

Parameters:	kind	– dataset type [steam / Kaggle]
	attribute	– grouping occurs according to this attribute
	partitions	– number of partitions of numeric data
Description:	returns a range of groups and the corresponding interval that the data belongs to	
Example:	/getGroupAndCount? attribute =xp_per_min& partitions =3	

Tab 6: Endpoint getGroupAndCount

7.7 getStages

host / getStats

Parameters:	kind	– dataset type [steam / Kaggle]
Description:	returns the number of rows and columns of the corresponding dataset	
Example:	/ getStats? kind = steam	

Tab 7: getStages endpoint

7.8 getSchema

host / getSchema

Parameters:	kind	– dataset type [steam / Kaggle]
Description:	returns the columns and corresponding type of the specified dataset	
Example:	/ getSchema? kind =steam	

Tab 8: getSchema endpoint

7.9 getDoubleGroup

host / getDoubleGroup

Parameters:	kind	– dataset type [steam / Kaggle]
	col1	- first column
	col2	- second column
Description:	returns grouping and counting by corresponding columns	
Example:	/ getDoubleGroup? kind =steam& col1 =leaver_status& col2 =radiant_win	

Tab 9: getDoubleGroup endpoint

7.10 getClusterStats

host / getClusterStats

Parameters:	
Description:	Returns the center points of the data for each cluster, i.e. information for each attribute of the data set.
Example:	/getClusterStats

Tab 10: getClusterStats endpoint

7.11 getClusterCount

host / getClusterCount

Parameters:	
Description:	Returns the number of data points for each cluster
Example:	/getClusterCount

Tab 11: Endpoint getClusterCount

7.12 postCluster

host / getClusterCount

Parameters:	gold	– money
	gold_per_min	– money per minute
	xp_per_min	– experience per minute
	kills	– kills during the game
	deaths	– deaths during the game
	assists	
	denials	– denials of murders
	last_hits	– final hit for killing
	hero_damage	– damage to opponents
	hero_healing	– team support
	tower_damage	– damage to towers
	level	– levels
Description:	Returns the corresponding attributes along with a new attribute indicating the type of clustering to which the data set belongs.	
Example:	/postCluster? gold =2000.0& gold_per_min =5113.0& xp_per_min =1231.0& kills =21.2 & deaths =3.3& assists =10.0& denials =34.3& last_hits =411.1 & hero_damage =56000.2& hero_healing =0.0& tower_damage =22000.0 & level =25.0	

8 References

[1] P. Norvig and SJ Russell, Artificial Intelligence: A Modern Approach, Harlow, United Kingdom: Prentice Hall, 2009.

[2] P. Norvig and SJ Russell, «Artificial Intelligence: A Modern Approach,» in *Artificial Intelligence: A Modern Approach* , Harlow, United Kingdom, Prentice Hall, 2019, p. 37.

[3] P. Norvig and SJ Russell, «Artificial Intelligence: A Modern Approach,» in *Artificial Intelligence: A Modern Approach* , Harlow, United Kingdom, Prentice Hall, 2009, p. 2.

[4] J. Han, M. Kamber and JP, Data Mining: Concepts and Techniques, Elsevier, 2020.

[5] I. Dabbura, «K-Means Clustering Algorithm Applications,» 17 September 2018. [Online]. Available: https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a. [Accessed 11 11 2019].

[6] DC «Mean Shift,» [Online]. Available: https://courses.csail.mit.edu/6.869/handouts/PAMIMeanshift.pdf. [Accessed 11 11 2019].

[7] S. Devs, "Spark Clustering," [Online]. Available: https://spark.apache.org/docs/latest/ml-clustering.html. [Accessed 13 11 2019].

[8] A. Agarwala and M. Pearce, "Learning Dota 2 Team Compositions," [Online]. Available: http://cs229.stanford.edu/proj2014/Atish%20Agarwala,%20Michael%20Pearce,%20Learning%20Dota%202%20Team%20Compositions.pdf. [Accessed 15 11 2019].

[9] Wikipedia, "Kaggle," [Online]. Available: https://en.wikipedia.org/wiki/Kaggle. [Accessed 12 11 2019].

[10] P. Bugnion, PR Nicolas and A. Kozlov, Scala: Applied Machine Learning, Packt Publishing, 2017.

[11] K. Rimple, A Gentle Intro to Docker for Developers, Chariot Solutions, 2019.

[12] Google, «Google Cloud Platform,» Google, 2017. [Online]. Available: https://cloud.google.com/. [Accessed 29 11 2019].

[13] D. Devs, «Dard Documentation,» 10-12 October 2011. [Online]. Available: https://dart.dev/guides. [Accessed 23 10 2019].

[14] T. Bock, «What is a Correlation Matrix,» [Online]. Available: https://www.displayr.com/what-is-a-correlation-matrix/. [Accessed 24 11 2019].