UNIVERSITY OF BARI
DEPARTMENT OF COMPUTER SCIENCE

Rocket League

Student name: *Pasquale De Marinis*

Course: *Big Data* – Professor: *Michelangelo Ceci*
Date: *April 6, 2022*

Contents

1. Introduction

This case study deals with the practical application of the Cross Industry Standard Process for Data Mining, in short CRISP-DM. The main objective is the analysis of Rocket League match replays in order to predict the skill of each player in a specific match.

This documentation will follow the six phases of CRISP-DM, thus in each section, a phase will be described, however the process is iterative, therefore it will anticipate in each section analysis that can be chronologically distant.

2. Business understanding

In this step the objective is to determine and understand the context and the scope of the project. So it has a lot in common with the initial steps of any significant project undertaking.

2.1. Determine Business Objectives

Figure 1: One of the main Rocket League artworks

In recent years video games gained a lot of popularity, above all, multiplayer games; some of this are considered competitive and many sportive events take place around them. Rocket League is a multiplayer competitive online game which is an hybrid that unifies soccer with car race games.

Thus, each match follows more or less soccer rules, but instead of human players there are cars controlled by the players. The number of players per team can be 1 (duel), 2, or 3 (standard). In our study we will focus on the main modality, that is 3vs3.

2.1.1. Ranking System

Figure 2: Rocket league rankings

In each modality, the player, after playing 10 competitive- online matches, is classified into a rank, based on the result of the 10 played matches. The ranks (visually shown in figure 2) are:

- Bronze I
- Bronze II
- Bronze III
- Silver I
- Silver II
- Silver III
- Gold I
- Gold II
- Gold III
- Platinum I
- Platinum II
- Platinum III
- Diamond I
- Diamond II
- Diamond III
- Champion I
- Champion II
- Champion III
- Grand Champion I
- Grand Champion II
- Grand Champion III
- Supersonic Legend

Each rank, is split in four divisions (Division I, II, III and IV). And, each rank with its divisions is itself a discretization of a number, that is the Match Making Ranking (MMR).

Tier	Division I	Division II	Division III	Division IV
Supersonic Legend	1.861 —	—	—	—
Grand Champion III	1.709 — 1.739	1.745 — 1.773	1.788 — 1.814	1.832 — 1.860
Grand Champion II	1.575 — 1.598	1.600 — 1.636	1.638 — 1.660	1.677 — 1.701
Grand Champion I	1.435 — 1.458	1.460 — 1.494	1.498 — 1.527	1.537 — 1.559
Champion III	1.315 — 1.333	1.335 — 1.366	1.368 — 1.394	1.402 — 1.420
Champion II	1.195 — 1.213	1.215 — 1.246	1.248 — 1.275	1.282 — 1.300
Champion I	1.075 — 1.093	1.095 — 1.127	1.128 — 1.148	1.162 — 1.180
Diamond III	995 — 1.003	1.004 — 1.027	1.028 — 1.051	1.052 — 1.060
Diamond II	915 — 923	924 — 947	948 — 971	972 — 980
Diamond I	835 — 843	844 — 867	868 — 891	892 — 900
Platinum III	773 — 778	779 — 797	798 — 816	817 — 825
Platinum II	715 — 718	719 — 737	738 — 756	757 — 765
Platinum I	655 — 658	659 — 677	678 — 696	697 — 705
Gold III	595 — 598	599 — 617	618 — 636	637 — 643
Gold II	535 — 538	539 — 557	558 — 576	577 — 585
Gold I	475 — 478	479 — 497	498 — 516	517 — 524
Silver III	415 — 418	419 — 437	438 — 456	457 — 467
Silver II	355 — 358	359 — 377	378 — 396	397 — 414
Silver I	295 — 298	299 — 317	318 — 336	337 — 354
Bronze III	235 — 238	239 — 257	258 — 276	277 — 290
Bronze II	172 — 178	179 — 197	198 — 216	217 — 233
Bronze I	0 — 118	121 — 137	140 — 155	157 — 172

Table 1: MMRs intervals related to each Division and Rank in Rocket League

The MMR goes from 0 to potentially infinity, however the last rank: Supersonic Legend, takes the interval [1861, +Infinity]. A detailed view on MMRs and rankings is shown in table 1.

When the player starts playing, it is assigned an MMR of 600, that correspond to Gold III Division II. The MMR increase or decrease basing on the outcome of the matches it plays, obviously if he wins it will increase, viceversa if he loses. How much it can gain or lose at the end of the match is based on the number of played matches. In the first match, the gain/loss is +/-150. In the second match the values is halved, and after, it logarithmically decrease, until after some dozen of matches, it converge to +/- 10.

With this system, the player, will be classified in its most appropriate rank after a few dozen of matches and, therefore will compete with players of the same skill level.

Our business objective is, therefore:

- Analyze the statistics of each player of a match and predict its rank based on the performances that he shown

This work will result in a system able to measure the skill shown in a match for each player in a "short" amount of time, in order to be implemented in a real system.

2.2. Determine Data Mining Goals

The goal is a classification/regression task. There are 22 ranks, or we can possibly group by in 8, (bronze, silver, gold, platinum, diamond, champion, grand champion, SSL), therefore, it could a classification task. On the other hand we would lose the natural ordering of the classes,

The criteria will be based on RMSE and MAE, it will be accepted a MAE inferior to 2.0 and a RMSE inferior to 2.5. Another criteria is the inference time, that should be less than 0.5 seconds, measured on a CPU AMD Ryzen 2600x.

3. Data understanding

In this step the objective is to consider the available data, understand their properties, check its quality and explore it through statistical methods.

3.1. Collect initial data

The data is collected through a platform called *ballchasing.com* that allows players to download a plugin for the game that will automatically load their replays to to platform.

The replays are then viewable into a web based frontend and is possible to analyze different statistics from the replay. The platform provides also different HTTP API:

- **Get replay**: returns the binary replay file given the ID;
- **Get replay list**: returns a json involving summary of replays (including the ID to get the full replay) given different filters;
- **Get replay statistics**: returns a full detailed statistics of the game for each player given the replay ID;
- **Upload replay** used by the plugin to upload replays;
- **Delete replay**.

It is important to clarify that a replay **is not a video** file as usual, but a large binary file containing various metadata on the match and the players and a table in which are listed all the information necessary to review the match. Therefore, for each instant *(frame)* of the game, are listed position, direction and velocity vector of each player and the ball, and also other information of the inputs the players is giving (e.g. player is using boost, player is drifting, etc...).

Extracting features from the *raw replays* can be very challenging, and the amount of possible features is quite high. However, the main problem about raw replays is API limitations, as it is possible to download only 200 replays per hour. Instead, we can

download the *replay statistics*, as they are very detailed and provide stats about all the players, furthermore the limitation is 2000 per hour.

Thus, in this project, it was used the calculated stats from the APIs. Each downloaded replay is saved in a JSON file. These, are hierarchical files that contain also a lot of metadata and other useless for our task information about the match (e.g. stadium, car personalization of each player etc...). It was discarded this data and take the stats about the six players in the match, which then results in 6 rows for each replay.

Data was collected using *random sampling* from the date. The date is sampled from a range starting from August 2021 until February 2022, that's because are the dates of the two last *"Seasons"* of Rocket League. At the end of each season there is a soft reset of ranks: the MMR doesn't change but the players have to do again 10 matches where the MMR change at the end of the match is higher. It was decided to not consider older seasons because the *playstyle* and the distribution of players in Rocket League changes over time.

However, it is necessary to discard some player rows where the rank information was missing. This is due to the fact that the rank in that match wasn't determined yet.

3.2. Describe data and features

The resulting dataset counts 404951 rows and 85 features in our data, divided in five categories: Core Stats, Boost Stats, Movement Stats, Poistioning Stats and Demolitions Stats. Let's briefly describe them:

Core Stats

- **shots**: shots performed;
- **shot**_against: shots the players undergoes while he is acting as goalkeeper;
- **goals**: goals performed;
- **goals_against**: goals taken;
- **saves**: goals saved;
- **assists**: assists performed;
- **score**: score cumulated at the end of the match;
- **mvp**: tells if is the most valuable player in the match;
- **shooting_percentage**: shots compared to the teams shots;

Boost Stats

- **bpm**: boost consumption per minute;

- **bcpm**: boost collected per minute;
- **amount_collected_big**: amount of boost collected with big pads. Boost can be collected through big or mini pads, big ones are at the edges of the playing field, and fulls the boost, mini ones are scattered trough the field and refill only 13%;
- **amount_collected_small**: amount of boost collected with mini pads;
- **amount_stolen_big**: amount of boost stolen to other players from big pads;
- **amount_stolen_small**: amount of boost stolen to other players from mini pads;
- **count_collected_big**: number of big pads taken;
- **count_collected_small**: number of mini pads taken;
- **amount_overfill**: amount of boost extra boost taken from big pads;
- **amount_overfill_stolen**: amount of boost extra boost taken from big pads to steal from other players;
- **amount_used_while_supersonic**: amount of boost used while in supersonic speed (useless as it is max speed);
- **time_zero_boost**: amount of time player has zero boost in the match;
- **time_boost_0_25**: amount of time player has boost between 0 and 25;
- **time_boost_25_50**: amount of time player has boost between 25 and 50;
- **time_boost_50_75**: amount of time player has boost between 50 and 75;
- **time_full_boost**: amount of time player has full boost;
- **percent_zero_boost**: percent of time player has zero boost in the match;
- **percent_boost_0_25**: percent of time player has boost between 0 and 25;
- **percent_boost_25_50**: percent of time player has boost between 25 and 50;
- **percent_boost_50_75**: percent of time player has boost between 50 and 75;
- **percent_full_boost**: amount of time player has full boost;
- **avg_amount**: average amount of boost in the match;

Movement Stats

- **avg_speed**;
- **avg_speed_percentage**: average speed compare to max speed;
- **total_distance**: total distance covered in the match;

- **time_slow_speed**: time spent moving slower than if you were boosting/dodging, i.e. just using throttle, less than 1400 uu/s;
- **time_boost_speed**: time spent moving at boost speed, e.g. while holding boost/dodging, $1400 + uu/s$, *uu* stands foru *unreal unit*, a distance measure unit in games made in Unreal Engine that equals $1cm$;
- **time_supersonic_speed**: time spent moving at supersonic speed, $2200 + uu/s$;
- **percent_slow_speed**: percentage time spent moving slower than if you were boosting/dodging, i.e. just using throttle, less than $1400uu/s$;
- **percent_boost_speed**: percentage of time spent moving at boost speed, e.g. while holding boost/dodging, $1400 + uu/s$;
- **percent_supersonic_speed**: percentage of time spent moving at supersonic speed, $2200 + uu/s$;
- **time_ground**: amount of time spent on the ground;
- **time_low_air**: amount of time spent in air but at low altitude w.r.t. the field;
- **time_high_air**: amount of time spent in air but at high altitude w.r.t. the field;
- **percent_ground**: percentage of time spent on the ground;
- **percent_low_air**: percentage of time spent in air but at low altitude w.r.t. the field;
- **percent_high_air**: percentage of time spent in air but at high altitude w.r.t. the field;
- **time_powerslide**: amount of time spent powersliding;
- **count_powerslide**: number of times player uses powerslide;
- **avg_powerslide_duration**;

Positioning Stats

- **avg_distance_to_ball**;
- **avg_distance_to_ball_possession**: average distance to the ball while possessing the ball;
- **avg_distance_to_ball_no_possession**: average distance to the ball while not possessing the ball;
- **avg_distance_to_mates**;
- **time_defensive_third**: amount of time player is in 1/3 of the field where his net is;

- **time_neutral_third**: amount of time player is in the neutral third;
- **time_offensive_third**: amount of time player is in 1/3 of the field where the opponent net is;
- **percent_defensive_third**: percentage of time player is in 1/3 of the field where his net is;
- **percent_neutral_third**: percentage of time player is in the neutral third;
- **percent_offensive_third**: percentage of time player is in 1/3 of the field where the opponent net is;
- **time_offensive_half**: amount of time player is in opponent half;
- **time_defensive_half**: amount of time player is in his team half;
- **percent_offensive_half**: percentage of time player is in opponent half;
- **percent_defensive_half**: percentage of time player is in his team half;
- **time_behind_ball**: amount of time the player is closer to its net than the ball;
- **time_infront_ball**: amount of time the player is closer to the opponent net than the ball;
- **percent_behind_ball**: percentage of time the player is closer to its net than the ball;
- **percent_infront_ball**: percentage of time the player is closer to the opponent net than the ball;
- **time_farthest_from_ball**: amount of time he is the player farthest from ball;
- **time_closest_from_ball**: amount of time he is the player closest from ball;
- **percent_farthest_from_ball**: percentage of time he is the player farthest from ball;
- **percent_closest_from_ball**: percentage of time he is the player closest from ball;
- **time_most_back**: amount of time he was the last defender of its team;
- **time_most_forward**: amount of time he was the first attacker of its team;
- **percentage_most_back**: amount of time he was the last defender of its team;
- **percentage_most_forward**: amount of time he was the first attacker of its team;
- **goals_against_while_last_defender**;

Demolitions Stats

- **inflicted**: demolitions inflicted to opponent players;

- **taken**: demolitions taken by opponent players.

We can right away notice that there are a lot of features correlated, all the time / percent features are a repetition and we could keep only one among the twos.

Among these we can distinguish **categorical** and **numeric features** (table 2):

Categorical	
mvp	

Dicsrete	
saves	goals_against_while_last_defender
assists	shots
goals	inflicted
taken	

Continuous	
percent_closest_to_ball	time_offensive_half
avg_speed_percentage	percent_boost_25_50
amount_collected_small	amount_collected
bcpm	percent_farthest_from_ball
avg_distance_to_mates	percent_high_air
percent_slow_speed	time_behind_ball
time_supersonic_speed	percent_offensive_half
count_powerslide	time_zero_boost
percent_ground	shooting_percentage
count_stolen_small	amount_stolen_small
avg_speed	percent_behind_ball
percent_zero_boost	avg_powerslide_duration
time_boost_50_75	amount_stolen_big
goals_against	percent_defensive_half
time_most_back	percent_defensive_third
time_farthest_from_ball	percent_low_air
count_collected_small	time_full_boost
time_offensive_third	time_defensive_half
amount_collected_big	count_collected_big
time_high_air	amount_stolen
time_defensive_third	shots_against
avg_amount	time_neutral_third
time_slow_speed	percent_most_forward
percent_neutral_third	avg_distance_to_ball_no_possession
time_closest_to_ball	total_distance
percent_boost_0_25	time_low_air
time_infront_ball	avg_distance_to_ball
score	time_boost_25_50
time_boost_0_25	time_boost_speed
amount_overfill	time_boost_75_100
bpm	time_ground
avg_distance_to_ball_possession	time_powerslide
percent_boost_50_75	percent_boost_75_100
percent_most_back	percent_boost_speed
count_stolen_big	amount_used_while_supersonic

percent_offensive_third	percent_supersonic_speed
amount_overfill_stolen	time_most_forward
percent_full_boost	percent_infront_ball

Table 2: Categorical and discrete feature disctinction

3.3. Verify data quality

All of our data is automatically collected, thus, is not subject to human errors. We cannot ensure that data is *accurate*, so that stored value is the actual value, but we can assume it always because of the collection process.

Data is **incomplete**. There where a bunch of missing data in some dozen of rows in columns:*percent_closest_to_ball percent_farthest_from_ball*, *percent_most_forward*, *avg_distance_to_mates*, *time_most_back*.
Missing values in this case are originated from leaving the match before the end, causing errors in the calculation of the metrics. We can notice that most of the missing values are in Bronze ranks, because they tend to care less about their rank and the game in general (leaving a ranked match will count as a loss and will ban you for short period from the ranked matches).

Data is **consistent**: time measurements are given in seconds, distances in *unreal units (= centimeters)* speeds in *unreal units per second*; then we have counts and percentages.

Data is **up-to date**: is collected from replays of the two last seasons, and, until the game mechanisms change, it isn't obsolete.

3.4. Clean data

In order to calculate summary statistics on our data we need to get rid of missing values and errors in our data. In this case the most appropriate solution is also the most simple, as the number of rows containing missing values are 199, thus, removing this rows from the dataset.

3.5. Data exploration

Let's see some statistics from our data:

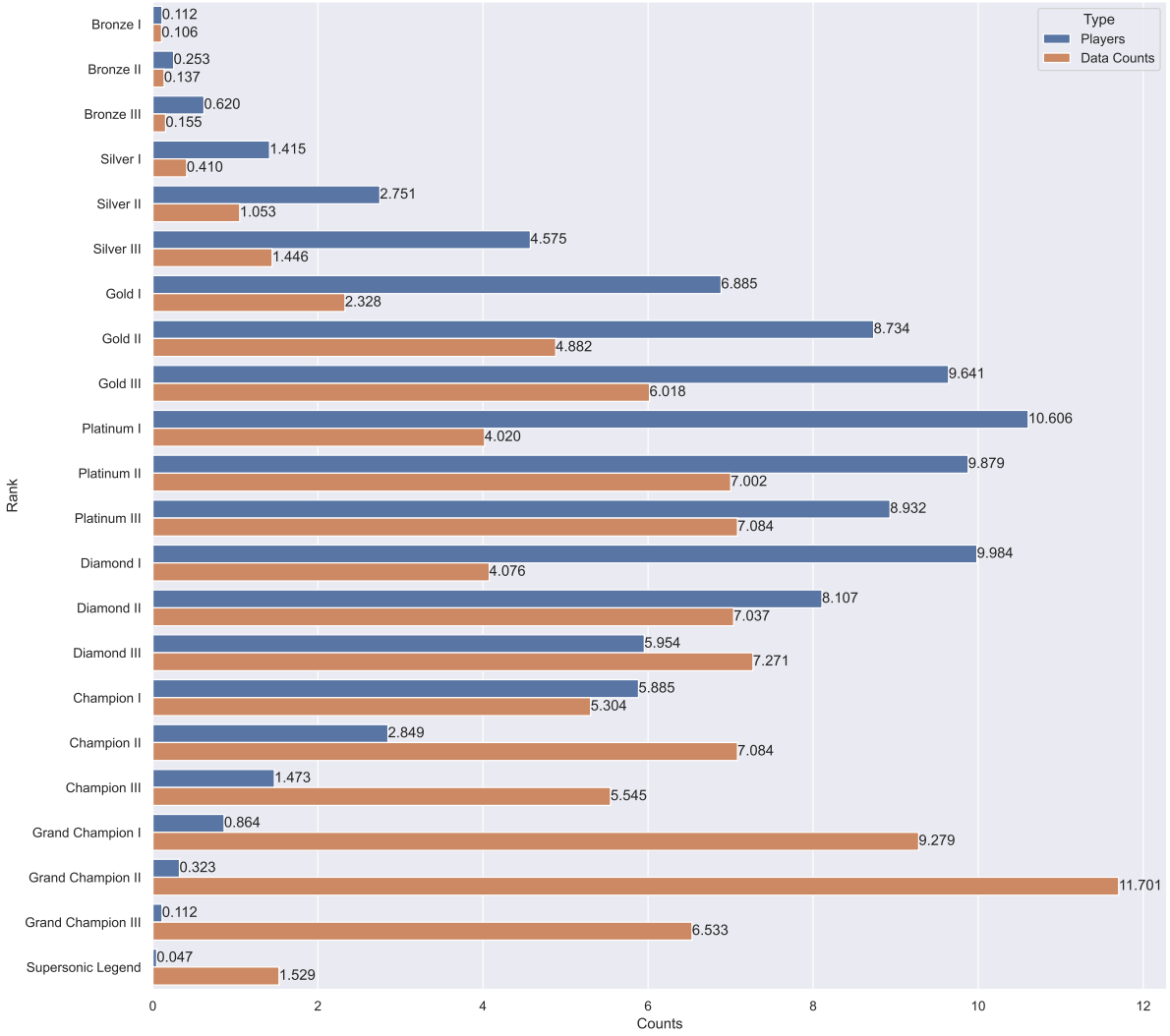

Figure 3: Distribution of players ranks (blue) vs. distribution of ranks in data (orange)

We can notice that the the bronze class had very few examples. This has two main reasons, the first one is that there are actually very few players in bronze, it's very difficult to get there basing on how ranking system works. The second reason is that players in bronze are casuals players that even don't know about the plugin necessary to upload replays. We can see the distribution of ranks for the current season (Season 5) in figure 3.5 (orange). In the same figure we can see in blue the actual distribution of players, that is more or less normally distributed and centered. The data distribution is instead shifted to the right; as we said, skilled players tends to use and know about the plugin.

4. Data preparation

Data selection was automatically performed during the collection of the dataset. As already said, replays are randomly sampled between two dates representing two Rocket League seasons.

For feature selection the situation is different, the number of features is quite high and we can early see that some of the are correlated, thus we plot a correlation matrix table 4:

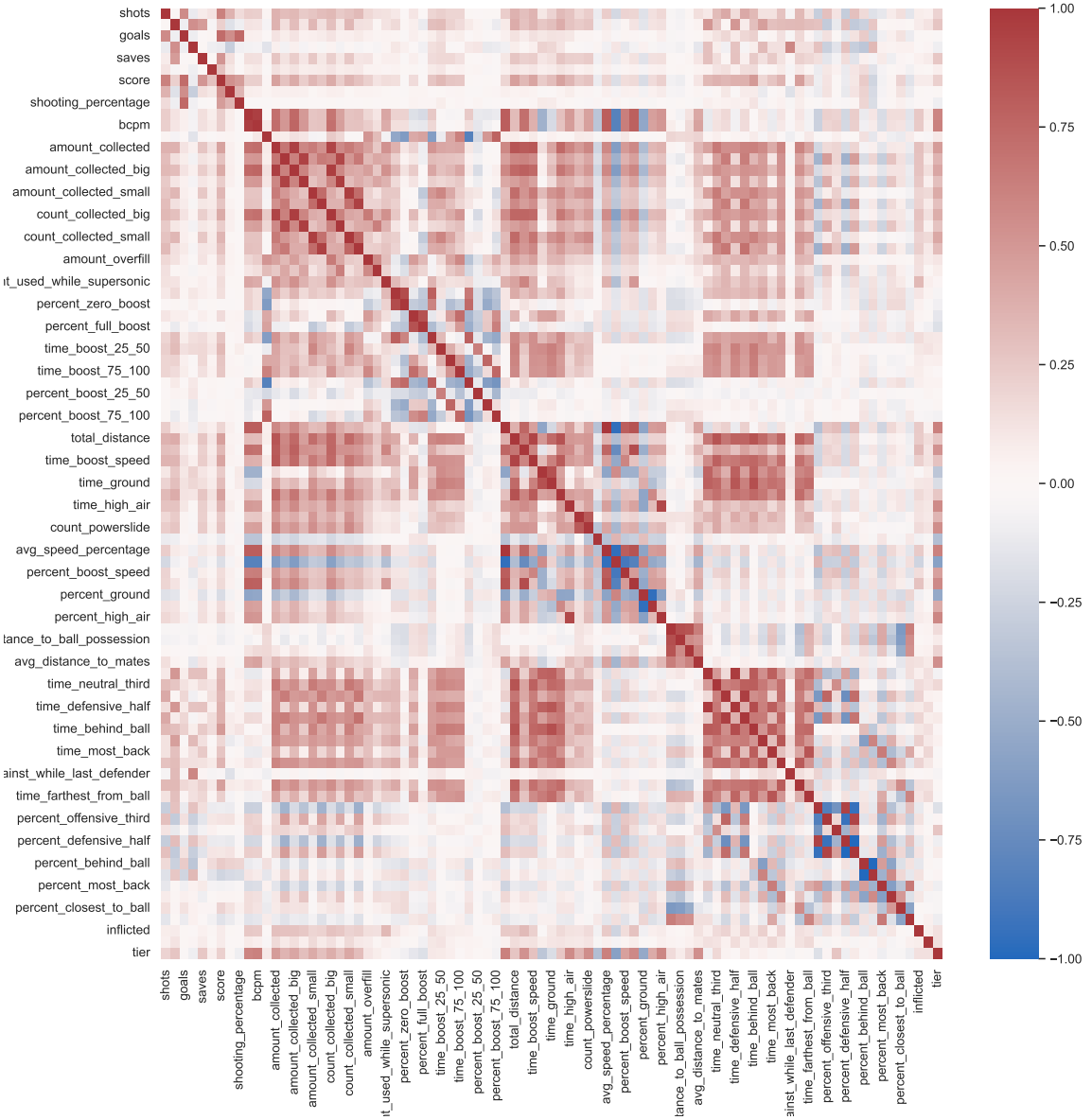

Figure 4: Features correlation matrix

Deleted feature	Correlated feature	Motivation
time_offensive_half	time_offensive_third	trivial
time_defensive_half	time_defensive_third	trivial
time_behind_ball	time_infront_ball	trivial
time_infront_ball	percent_infront_ball	trivial
time_ground	time_low_air, time_high_air	sum up to 1
time_slow_speed	time_boost_speed, time_supersonic_speed	sum up to 1
time_most_back	time_most_forward	trivial
time_most_forward	time_infront_ball	player whos most forward is usually infront of the ball
percent_defensive_half	percent_offensive_half	trivial
percent_behind_ball	percent_infront_ball	trivial
avg_speed	percent_slow_speed	trivial
percent_ground	percent_low_air, percent_high_air	sum up to 1
time_farthest_from_ball	time_defensive_third	defender is most of the time farthest from ball
percent_boost_75_100	percent_boost_X_Y	sum up to 1
time_boost_0_25	percent_boost_0_25	trivial
time_boost_25_50	percent_boost_25_50	trivial
time_boost_50_75	percent_boost_50_75	trivial
time_boost_75_100	percent_boost_75_100	trivial
time_defensive_third	percent_defensive_third	trivial
time_offensive_third	percent_offensive_third	trivial
percent_offensive_half	percent_offensive_third	Third is more specific than half
percent_defensive_half	percent_offensive_third	Third is more specific than half
percent_defensive_third	percent_offensive_third percent_neutral_third	sum up to 1
percent_slow_speed	percent_boost_speed	sum up to 1
avg_amount	percent_boost	trivial
avg_speed_percentage	avg_speed	trivial
time_defensive_half	percent_defensive_half	trivial
time_offensive_half	percent_offensive_half	trivial
count_collected_big	amount_collected_big	trivial
count_collected_small	amount_collected_small	trivial
count_stolen_big	amount_stolen_big	trivial
count_stolen_small	amount_stolen_small	trivial
time_low_air	time_boost_speed	When jumping in low air players usually use boost
time_high_air	percent_high_air	trivial
time_supersonic_speed	percent_supersonic_speed	trivial
time_zero_boost	percent_zero_boost	trivial
time_full_boost	percent_full_boost	trivial
amount_stolen	amount_stolen_big	trivial
amount_collected	amount_collected_big	trivial
avg_distance_to_ball	avg_distance_to_ball_possession	avg_distance_to_ball is more general than avg_distance_to_ball_possession
total_distance	time_boost_speed	The fast player is the more it travels
time_powerslide	count_powerslide	trivial
bcpm	bpm	Boost consumption is directly connected to boost usage

Table 3: List of deleted features and the correlated ones

By this selection we removed 42 features, thus, nearly halving the dataset vertically. The adopted criterion is based on logical correlation, thus, by inspecting correlations, the removed features are the ones where we can find a motivation for the correlation, otherwise the correlation can be spurious.

In order to further reduce the dimensionality of the dataset another two tests have been ran: the information gain test and the chi square test, results are shown in figure 4 and figure 4:

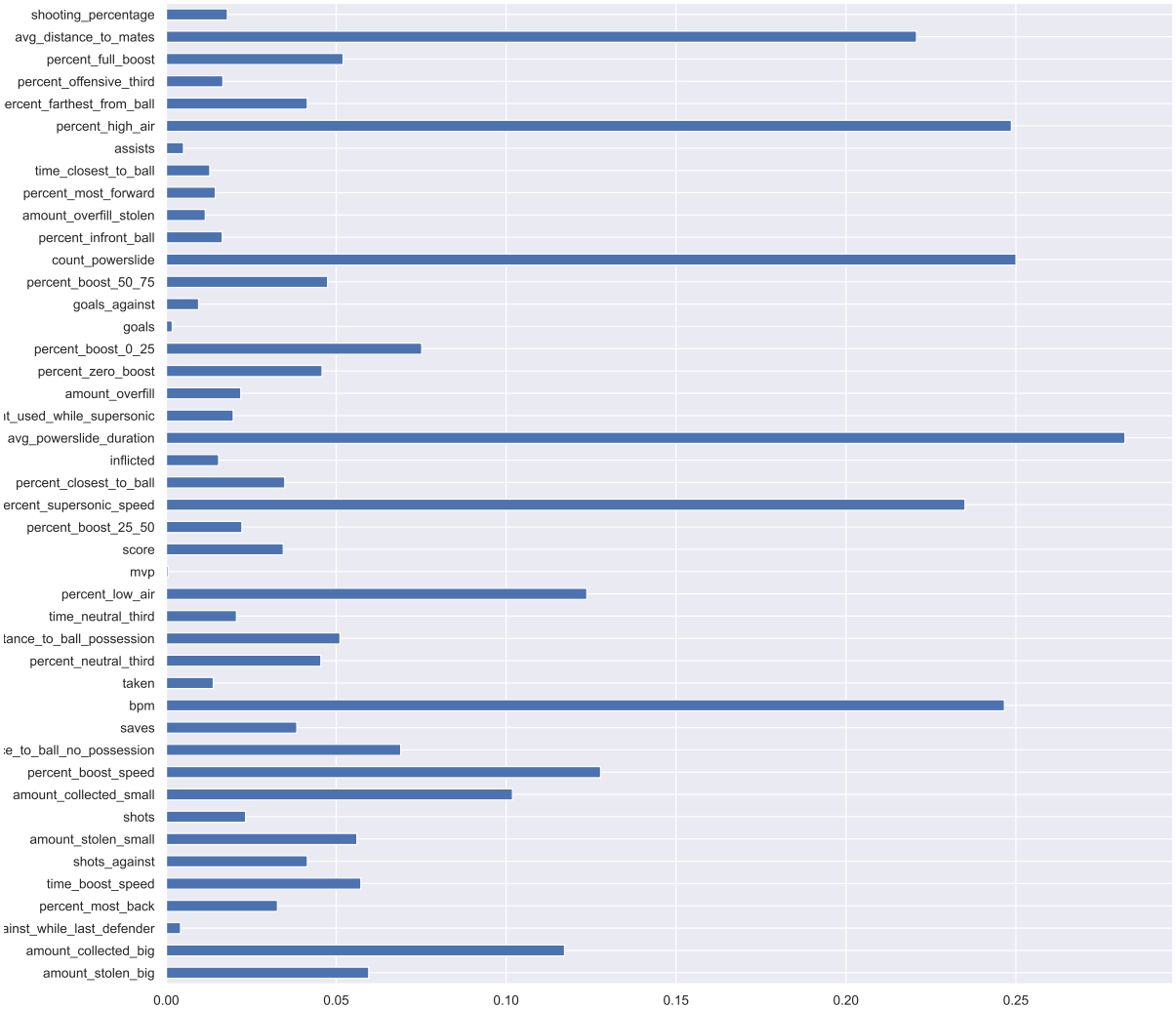

Figure 5: Information gain results

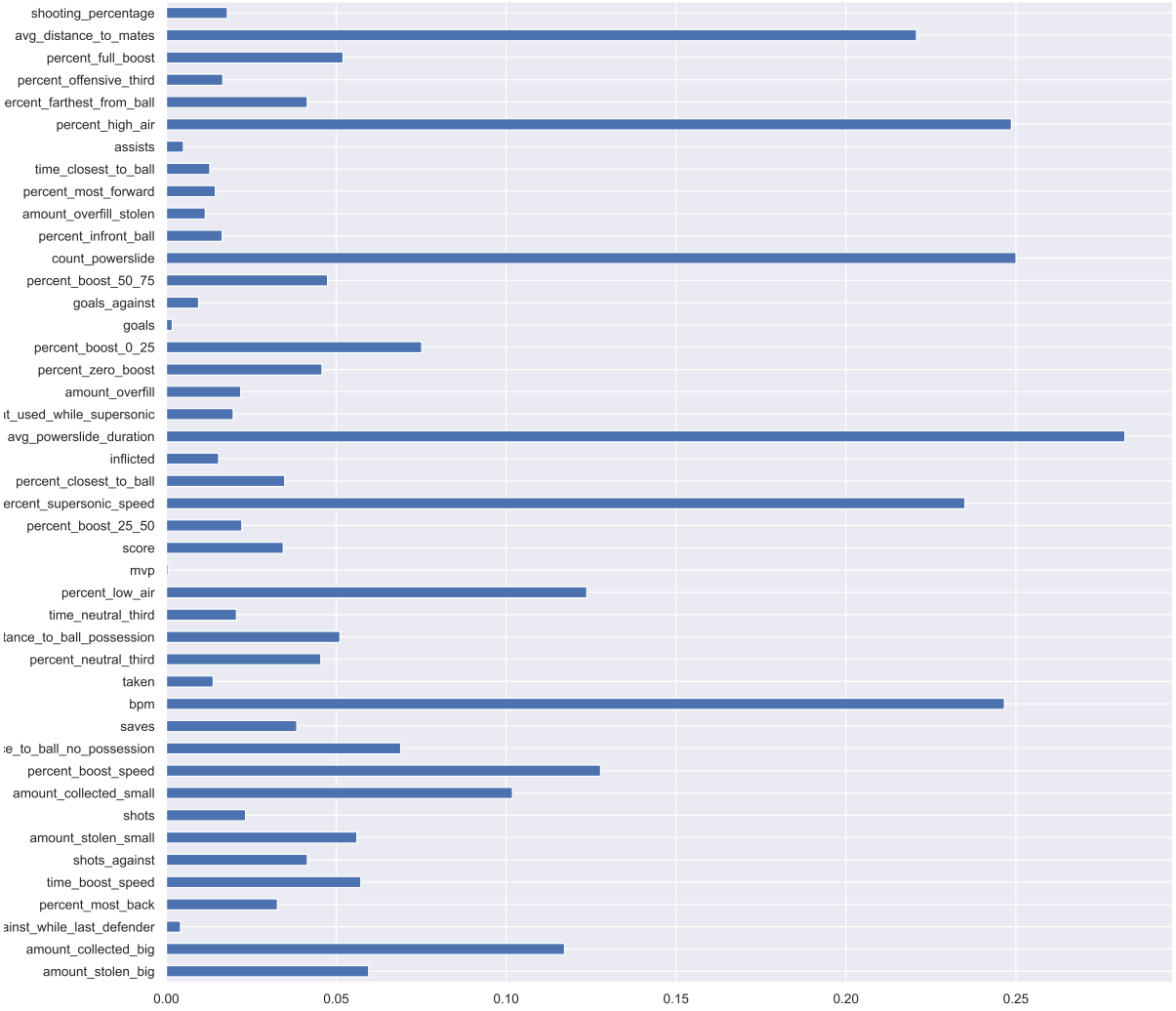

Figure 6: Chi square test results

The results gave another insight on some kind of features uninformative. The features that are strictly related to the specific match and not on the playstyle and skill of the player. Such features are *goals*, *mvp*, *assists* and *goals_against_while_last_defender*.

4.1. Data construction

In this step there will be gonna built the final dataset provided to the models.

Features of this dataset have different measurement units and scales, therefore features with bigger scales will be considered more important for some models, for example linear regression. We need to scale the data. The used scaling technique is called *Robust Scaling*. It subtract the median and divides using the *interquantile range (IQR)* instead of the mean and the standard deviation differently from normalization. It has been used the IQR between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile). This procedure has been applied to all the features except for some groups that needed particular attention. The first groups are the **percentual** features, they range from 0 to 100, therefore the normalization is useless; they have been simply divided by 100 to make them range from 0 to 1.

Other groups are:

- *avg_distances*: which includes distance to ball with and without possession and distance to mates
- *overfill*: which includes amount of boost overfill, amount overfill stolen and stolen with big pads

These groups of features have relations and constraints that can be removed by the normalization, for example amount_overfill is always greater than amount_overfill_stolen, and their difference is important. Therefore, each group, has been standardized like it is a unique features, in order to keep the relations

4.1.1. Factor analysis

Factor analysis is a linear statistical model. It is used to explain the variance among the observed variable and condense a set of the observed variable into the unobserved variable called factors. Observed variables are modeled as a linear combination of factors and error terms (Source). Factor or latent variable is associated with multiple observed variables, who have common patterns of responses. Each factor explains a particular amount of variance in the observed variables. It helps in data interpretations by reducing the number of variables. Factor Analysis addresses the problem of analyzing the structure of the interrelationships, among a large number of variables by defining a set of common underlying dimensions, known as **factors**. In this case, the analysis is suitable, and could improve the final metrics of the classifiers.

Firstly, it is ran a statistical test, the **Bartlett test of Sphericity**; the result of the *p-value* is 0.0, which is below the significance level, that is 0.01, this means that our data is suitable for PCA or factor analysis, in other terms, the correlation matrix is not an identity matrix, therefore there are variables that can be compressed because are correlated. After that, it another statistical test, the **Kaiser-Meyer-Olin** test. KMO estimates the proportion of variance among all the observed variables. Lower proportion is more suitable for factor analysis. KMO values range between 0 and 1. Value of KMO less than 0.6 is considered inadequate. The resulting value is 0.74, which is not excellent but still good to perform factor analysis.

The next step is choosing the number of factors, it is possible to use the Kaiser criterion or the scree plot.

Listing 1: Eigenvalues vector

```
8.23560479, 4.23322434, 3.22683551, 2.81685919, 2.30287244,
1.89468945, 1.77176288, 1.32000825, 1.24627339, 1.00624468,
0.96403381, 0.92449632, 0.84269987, 0.82747457, 0.77226586,
0.76199741, 0.69723765, 0.69105217, 0.67808479, 0.60368556,
0.59190556, 0.57496769, 0.51825302, 0.46937353, 0.37548574,
0.34613834, 0.32115293, 0.3107348 , 0.27012021, 0.2462839 ,
0.22761797, 0.18568548, 0.1833516 , 0.14390633, 0.12192532,
0.11567416, 0.08087169, 0.03868107, 0.02639674, 0.02272699,
0.01134398
```

We can see listing 1, that there are only 8 values above 1.

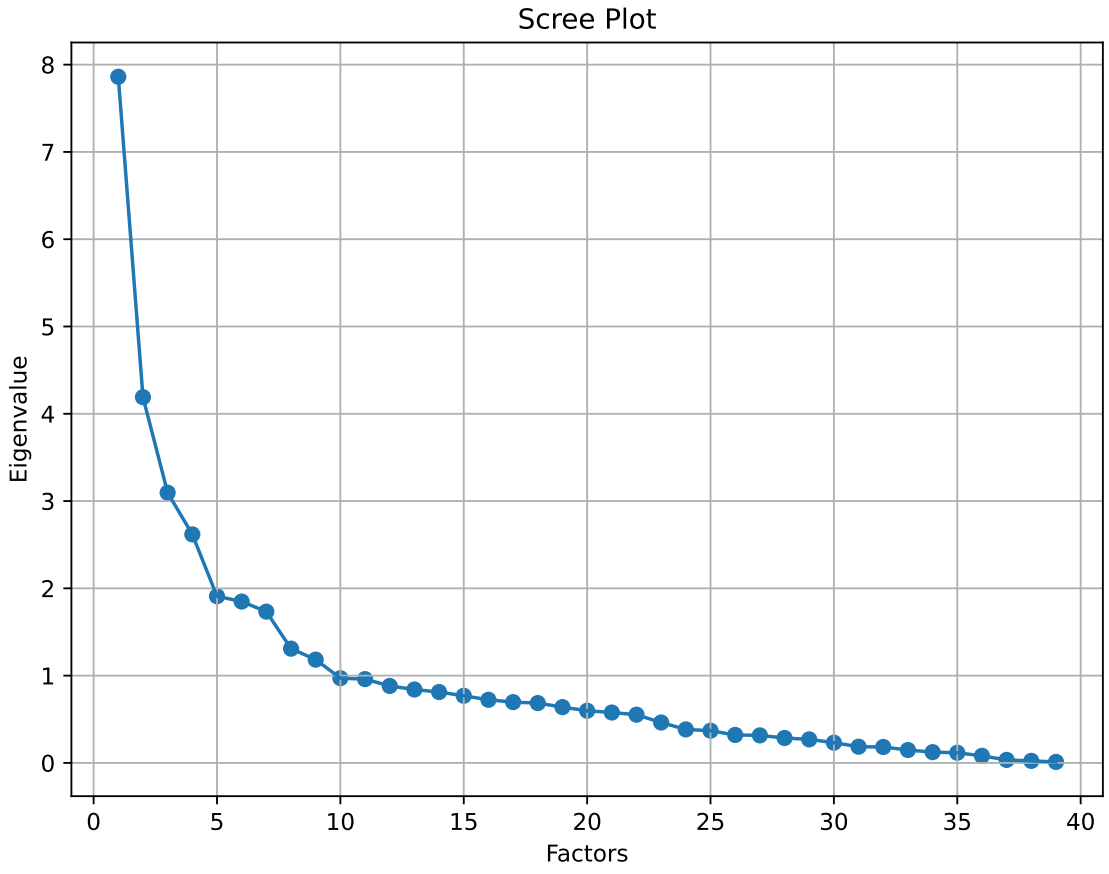

Figure 7: Scree plot

Also from the scree plot figure 4.1.1, the elbow is around 8, so we can state the 8 factors should be adequate. So we perform the factor analysis and print the *factor loadings* figure 4.1.1.

	time_closest_to_ball	time_infront_ball	amount_stolen_small	time_boost_speed	amount_collected_small	time_neutral_third
0	0.600782	0.625109	0.678427	0.751107	0.815739	0.885798

	count_powerslide	avg_distance_to_mates	amount_collected_big	percent_low_air	percent_high_air	percent_boost_speed	percent_supersonic_speed	bpm
1	0.518287	0.524817	0.529616	0.551254	0.576132	0.628194	0.726228	0.844565

	percent_closest_to_ball	percent_farthest_from_ball	avg_distance_to_ball_possession	avg_distance_to_ball_no_possession
2	-0.708902	0.721991	0.765407	0.776998

	percent_boost_50_75	percent_zero_boost	percent_boost_0_25
3	-0.602884	0.780163	0.911816

	amount_overfill	amount_used_while_supersonic	amount_collected_big
4	0.529966	0.580751	0.613159

	percent_most_back	time_infront_ball	percent_most_forward	percent_infront_ball
5	-0.612643	0.501344	0.723758	0.755299

	goals_against	shots_against
6	0.565508	0.824209

	shots	score
7	0.501795	0.949000

Figure 8: Loadings for each factor that are greater than 0.5 in absolute value

For sure we can say that variable in factor 6 and 7 are related and variables in factor 3 are all percentage of boost. For factor 5 and 2, we have two groups of related variables. Variables in factor 4 are all amount of boosts. Lastly, for factors 0 and 1 there are some related variables, but others doesn't have a clear interpretation of their relations.

4.2. Final produced datasets

From the preparation phase we produced 5 datasets:

- Dataset preprocessed: the dataset produced by the scaling;
- Dataset grouped preprocessed: the dataset produced by scaling done with grouping some variables;
- Dataset merged factored: the dataset produced by performing a factor analysis

5. Modeling

In this phase, data mining models are chosen and an evaluation plan is designed. After that, models are built, trained and evaluated.

5.1. Select modeling technique

According to section 2.2, our task is regression, thus *supervised learning*. Also, it is necessary to consider that there are numerical features. From the plethora of models, has been chosen firstly Logistic Regression, according to Occam razor, always start from simpler models. Then have been select others that from literature, are known as accurate models, such as, Naive Bayes, k-Nearest Neighbors, Random forest, XGBoost and Multi Layer Perceptron. In particular, for Naive Bayes, it was used the Gaussian Naive Bayes, suitable for numeric features.

5.2. Generate Test Design

After selecting the models, it is crucial to chose appropriate technique for assessing the performances of the models.

5.2.1. Metrics

First of all, it is needed to chose the metrics to calculate the performances. In this case, having a regression, the most used metrics are *Mean Absolute Error (MAE)* and *Root Mean Squared error (RMSE)*. The guidance metric to choose models will be RMSE, in order to penalize more greater errors.

5.2.2. Evaluation technique

The dataset is firstly split into training and test, 2/3 for the training and 1/3 for the test, so the method used is the *holdout*. As validation technique, K-fold cross validation is chosen for parameter selection, with K=10, as is a popular number of folds in literature. Thus, for each model, after the best parameters have been found, it is retrained on the whole training set and then tested. Having 404951 rows, means that we be trained on 269968 examples and test on 269967. In the validation, each fold will be of 13498 samples.

5.3. Build model

In this section there will be reported the results for each model, and each preprocessed dataset. Grid search has been performed over all the hyperparameters chosen.

5.3.1. Dummy Regressor

A random regressor has been tested to check the gain of the other regressors. Two strategy have been applied: **Median** and **Mean**. Results in table 4. Note that results on different preprocessing are obviously the same.

strategy	fit time	score time	MAE	RMSE
median	0.1383	0.0047	3.8826	4.5579
mean	0.1469	0.0026	3.9039	4.5510

Table 4: Validation results of the dummy regressor

5.3.2. KNN Classifier

K-Nearest Neighbors classifier, with two different K. Results in table 5

preprocess	n° neighbors	fit time	score time	MAE	RMSE
normal	5	0.0596	256.0697	2.1037	**2.7212**
	3	0.0536	235.6941	2.1987	2.8566
grouped	5	0.0585	253.4615	2.1245	2.746
	3	0.0579	232.959	2.2201	2.8845
factored	5	0.4768	6.0232	2.1717	2.8005
	3	0.4712	5.9163	2.2761	2.9521

Table 5: Validation results of KNN classifier

5.3.3. Linear classifier

Attempt to fit a simple model for this task. Results in table 6

preprocess	fit time	score time	MAE	RMSE
normal	3.6218	0.0109	1.8208	2.3364
grouped	3.607	0.012	1.821	**2.3360**
factored	0.0579	232.959	2.2201	2.8845

Table 6: Validation results of linear regressor

5.3.4. Multi Layer Perceptron

MLP classifier, with batch size of 256 and 3 hidden layers, the first with 256 units, the second with 128 and the last with 64. Results in table 7

preprocess	alpha	learning rate	fit time	score time	MAE	RMSE
normal	1.00E-05	0.001	143.95	0.112	1.5548	2.025
	1.00E-05	0.01	121.33	0.1305	1.5644	2.0338
	0.001	0.001	240.77	0.2452	1.555	**2.0245**
	0.001	0.01	217.85	0.3003	1.5637	2.0329
	0.0001	0.001	139.1	0.129	1.5564	2.0256
	0.0001	0.01	195.78	0.3088	1.563	2.0312
grouped	1.00E-05	0.001	149.46	0.114	1.5533	**2.0225**
	1.00E-05	0.01	135.61	0.1441	1.5617	2.0322
	0.001	0.001	163.42	0.1768	1.5588	2.0273
	0.001	0.01	168.28	0.2991	1.5649	2.0342
	0.0001	0.001	144.42	0.1276	1.5548	2.023
	0.0001	0.01	154.14	0.2273	1.562	2.0312
factored	1.00E-05	0.001	87.886	0.1004	1.9505	2.5106
	1.00E-05	0.01	84.601	0.1082	1.9586	2.5181
	0.001	0.001	89.773	0.1131	1.9526	2.5106
	0.001	0.01	112.35	0.1692	1.9542	2.5154
	0.0001	0.001	96.882	0.1057	1.9473	**2.5097**
	0.0001	0.01	113.13	0.1467	1.955	2.5141

Table 7: Validation results of MLP regressor

5.3.5. Gaussian Naive Bayes

Gaussian Naive Bayes regressor, with a variable smoothing of 1e-09. Class priors given are the relative frequencies for each "classes". The term can be mislead as we are in practice making regression, but in this case we are building the predictor as a classifier. Results in table 8

preprocess	priors	fit time	score time	MAE	RMSE
normal	None	1.067	0.9716	2.4879	**3.5729**
	Class frequencies	1.0951	0.9856	2.4887	3.5748
grouped	None	1.0737	0.9685	2.4879	**3.5729**
	Class frequencies	1.0962	0.9611	2.4887	3.5748
factored	None	0.3195	0.2247	2.3821	**3.4159**
	Class frequencies	0.3609	0.2394	2.3828	3.4181

Table 8: Validation results of Gaussian Naive Bayes

5.3.6. Random forest

Random forest regressor. Results in table 9

preprocess	criterion	n° estimators	fit time	score time	MAE	RMSE
normal	poisson	100	2375.7	0.7736	2.281	3.0503
	poisson	10	298.89	0.1124	2.3585	3.1484
	squared error	100	149.54	0.2951	**1.6597**	2.1611
	squared error	10	18.873	0.0483	1.7472	2.2769
grouped	poisson	100	5853.4	1.8964	2.2811	3.0503
	poisson	10	582.26	0.2023	2.3608	3.1499
	squared error	100	378.69	0.5909	**1.6603**	2.1624
	squared error	10	39.668	0.0742	1.7486	2.2814
factored	poisson	100	778.53	0.7561	2.3259	3.0673
	poisson	10	98.381	0.1049	2.4045	3.166
	squared error	100	53.13	0.2947	**1.9749**	2.5454
	squared error	10	7.0736	0.0485	2.0652	2.6674

Table 9: Validation results of Random Forest regressor

5.3.7. XGBoost

XGBoost regressor. Results in table 10

preprocess	importance	n° estimators	fit time	score time	MAE	RMSE
normal	weight	100	30.784	0.0179	1.6236	**2.1006**
	weight	10	3.4488	0.0091	1.8586	2.3444
	cover	100	32.523	0.0282	1.6236	2.1006
	cover	10	3.7628	0.0138	1.8586	2.3444
	gain	100	34.136	0.0168	1.6236	2.1006
	gain	10	3.9071	0.013	1.8586	2.3444
grouped	weight	100	33.204	0.0228	1.6236	**2.1006**
	weight	10	3.6957	0.0165	1.8586	2.3444
	cover	100	33.28	0.0211	1.6236	2.1006
	cover	10	3.6767	0.0124	1.8586	2.3444
	gain	100	32.745	0.0175	1.6236	2.1006
	gain	10	3.4516	0.0119	1.8586	2.3444
factored	weight	100	17.309	0.0135	1.9855	**2.5531**
	weight	10	1.9088	0.0074	2.1279	2.6666
	cover	100	17.352	0.0141	1.9855	2.5531
	cover	10	1.9101	0.0077	2.1279	2.6666
	gain	100	18.898	0.017	1.9855	2.5531
	gain	10	2.0845	0.0063	2.1279	2.6666

Table 10: Validation results of XGBoost regressor

5.4. Model comparison

After the hyperparameters have been selected from the validation and the final models have been trained on the whole training set, their results have to be compared. We can view them in table 11. MLP regressor outperformed every other regressor with every preprocessing type. Despite its simplicity, the linear model is just below the MLP and had better results than most complex models such as XGBoost and RandomForests in a very short fit time. KNN, in addition to have poor results, has a long score time, therefore it isn't the most suitable for real time applications.

preprocess	model	fit time	score time	MAE	RMSE
normal	dummy	0.1382	0.0045	3.9039	4.5511
	knn	0.0676	296.0647	2.0999	2.7127
	linear	3.9348	0.0119	1.8253	2.351
	mlp	**256.14**	**0.2554**	**1.5572**	**2.0247**
	naive bayes	1.2735	1.0172	2.4843	3.562
	random forest	155.85	0.3898	1.6568	2.1576
	xgb	33.624	0.0185	1.6245	2.1024
grouped	dummy	0.1383	0.0047	3.9039	4.5511
	knn	0.0685	293.4415	2.1222	2.7418
	linear	3.917	0.0123	1.8253	2.351
	mlp	**163.33**	**0.1234**	**1.5626**	**2.0275**
	naive bayes	1.2484	1.1646	2.4843	3.562
	random forest	383.82	0.6383	1.6568	2.158
	xgb	34.464	0.0245	1.6245	2.1024
factored	dummy	0.1379	0.0039	3.9039	4.5511
	knn	0.4878	6.5152	2.1222	2.7418
	linear	0.394	0.0084	1.8253	2.351
	mlp	**91.344**	**0.1004**	**1.5642**	**2.0268**
	naive bayes	0.3593	0.2543	2.4843	3.562
	random forest	58.197	0.3494	1.6569	2.1572
	xgb	19.148	0.0156	1.6245	2.1024

Table 11: Testing results for each model

6. Evaluation

Recalling the data mining goal, an RMSE inferior to 2.5 and a MAE inferior to 2.0. Except KNN and Naive Bayes, all the other models satisfied the data mining goal, and the best performing one, reached an RMSE of 2.0247 and a MAE of 1.5572. With such an error, the model could be used in game at the end of the match to classify the players or to show how they performed during the game. The inference time is surprisingly low, considering the *score time*, that calculates the whole test set, thus, about 269967 rows, in many cases it is lower than success the criteria.

6.1. Review process

This work brought an acceptable error to implement a ranking system in Rocket League based on Machine Learning, but it may be improved by further exploring more complex Neural Network models, since the MLP was the best performing model.

Another more complex approach, is to extract features directly from the raw replay, instead of the given statistics. However, this would require more time to gather replays, given the limitations of the *ballchasing* APIs and a tricky feature extraction phase, that could be manual or automatic using suitable Neural Network models, such as Recurrent NNs, since the replay is a large temporal sequence.

7. Deployment and Maintenance planning

For this project, the deployment could be done directly in the actual game, by integrating the model. The computation can be done client-side as the resulting models have a low inference time. As already mentioned, the predictive model can substitute the initial phase of the current ranking system that requires 10 matches to rank a player. So, they could be reduced to 3 matches, in order to have more robust scores, and the ranking can be done by averaging the results. After the initial ranking the system can be used in symbiosis with the current one. The amounts of gained or loss points at the end of the match, based on the win/loss can be reduced or increased for each player on the basis of the score given by the system, also in relation with the score given to the other players in the match. Therefore, it could end up in a fairer ranking system.

There is another option of deployment, the system can be used to by players to measure how they performed in a given match, therefore, it could be set up a server accessible via API or even with a web front end, where players can load their replays in order to receive the score given by the system.

Playstyle in Rocket League changes over time, therefore there will be the data-drift problem; thus, the model should been retrained in order to maintain their performances, at least once every three months.

References