Master Degree in Artificial Intelligence

Machine Learning Project

Dota 2 Team Winner Prediction

Academic Year: 2024/2025

Enrico Loda - VR522872
Luca Panariello - VR518122

University of Verona
Department of Computer Science

Contents

1 Introduction

1.1 What is dota 2?

Dota 2 is a multiplayer online battle arena (MOBA) video game released in 2013. It is played by millions of people around the world and is a staple of esports. The game consists of matches between two teams of five players, with each team occupying and defending their own separate base on the map. Each of the ten players independently controls a character known as a hero, who has unique abilities and different play styles. During a match, players earn experience points (XP) and gold to purchase items for their heroes in order to defeat the opposing team's heroes in player-versus-player (PvP) combat. A team wins by being the first to destroy the opposing team's Ancient, a large, sturdy structure located in the centre of each base.

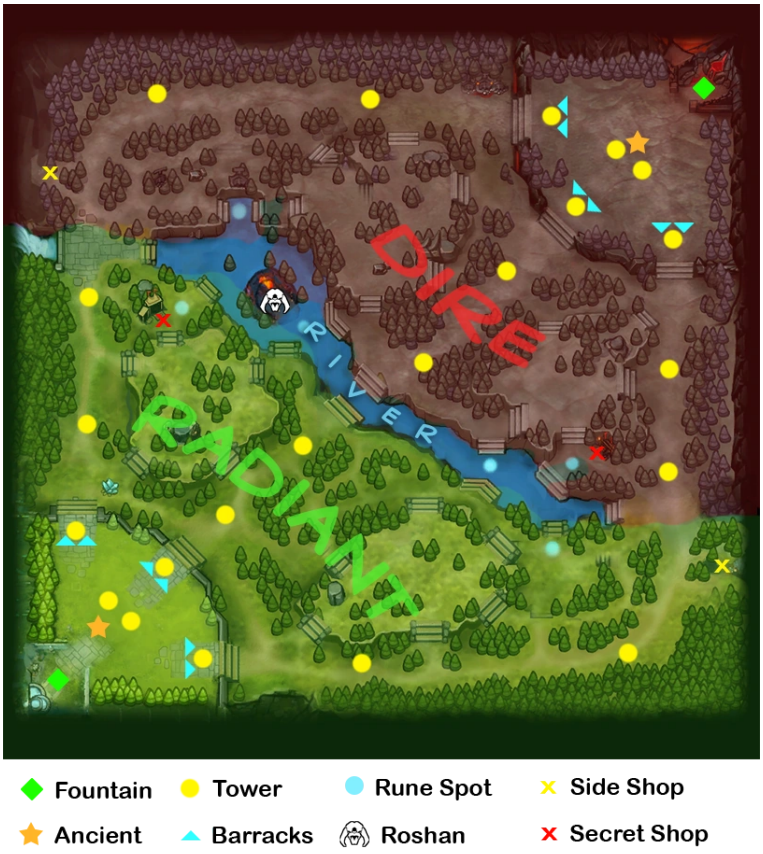


Figure 1: Dota Map

1.2 Motivation and rationale

Dota 2 is one of the most popular esports and one of the most lucrative, with prize pools of up to $40 million thanks to its prize pool funding system, which uses the sale of Dota TV tickets and bundles to increase the overall prize pool. Showing a prediction system of the winner during a competitive match, with the probability of both teams winning, is informative for the public and can increase their engagement in the competition.

Another possible implementation would be to introduce a "referee" type system into the game, which would recognise when a team is at a huge disadvantage due to the model's confidence in its predictions, and help the team in need in various ways.

1.3 Data Provenance

The dataset belongs to a competition hosted on Kaggle[1], the goal of which is to build a classifier model that predicts which team will win, given data extracted at one point during an ongoing match.

The metric used by the hosts to evaluate the performance of the models is the **Receiver Operating Characteristics - Area Under the Curve (ROC)** .

The dataset is provided in the form of 3 CSV files:

- `train_features.csv`
- `train_targets.csv`
- `test_features.csv`

The first two files were used for the training phase, while the last file is used to test the model and send the prediction in the form of probability confidence. According to the true results and the ROC-AUC score, a Kaggle score is given, along with the ranking on the public leaderboard.

The hosts have provided the raw data in an additional JSON file, along with CSV files containing processed features and target values. This allows users to extract additional relevant data if desired.

1.4 State of the Art

The main approach used for this type of task is the use of **Gradient Bosting** along with **Decision Trees** and **Logistic Regressors**.

Some neural networks were also used, but the best results were obtained with Gradient Boosting Models.

The model that achieved the best public score is a Gradient Boosting model using **XGBoost** or **LightGBM** library, combined with a heavy feature extraction process from the JSON file [2].

Another interesting project shows how it is possible to perform feature aggregation of the players stats using synthesis indices such as max, min and mean [3].

A further project performs a peculiar feature extraction, taking the positions of each player in the same team and drawing polygons with those positions as vertices. Area, intersection, perimeter and centroid are the new features that the author will work with [4].

1.5 Objectives

The objectives of this project are:

1. Create new type of dataset that the competition has not tried for this type of task.
2. Build different models to try out the new datasets and show that they are better than the original dataset and equivalent to other modified datasets, but with fewer features.
3. Confirm that for this particular classification task; information-based models, particularly GradientBoost, outperform other non-parametric models.
4. Achieve a minimum ROC-AUC score of 0.8 and a minimum Kaggle score of 0.8 on the competition's public ladder.

From the state of the art, this project is inspired by the the augmentation of individual player stats of the same team, but instead of using mean, max and min, the sum is used, which could be more informative than the other indices since each player contributes to the success of the match. Since the positions of the players cannot be summed up, a position mean or a similar approximation could be created, inspired by the polygon feature extraction project[4].

Gradient Boost is likely to perform better than the other models, so it will be used to see the maximum results from the new dataset.

2 Implementation

2.1 Dataset Analysis

The training set consists of 39675 samples with 246 features each. All existing features correspond to non-negative values.

The target file contains 6 target variables, that can be used for different tasks, in this project only the variable `"radiant_win"` has been considered, which is a boolean that denotes whether the radiant team has won or not (in this case the dire have won).

It can be noted that the dataset is actually well balanced with a percentage of 52.5% of wins for the radiant and 48.5% of losses. For this reason no resampling was done.

The features can be divided into two categories:

General Features, which describe the general direction of the game.

- `match_id_hash`: String. The alphanumeric identifier of the match. Id Feature.
- `game_time`: Float. Represents the time of the current game. Numerical Feature.
- `game_mode`: Integer. Corresponds to the type of game selected and may affect the selection and the pick order of heroes. Categorical Feature.
- `lobby_type`: Integer. Indicates the type of lobby between public, ranked and others. Categorical Feature.
- `objectives_len`: Integer. The length of the milestone achieved during the match, such as Roshan (main boss) killed and towers destroyed. Numerical Feature
- `chat_len`: Integer. The number of messages in the match chat. Numerical Feature.

In total there are 6 general features.

Player Features, which describe the stats of each player belonging to one of the two teams *Radiant* `r` or *Dire* `d`. There are 5 players per team, so each player feature starts with the team `t` and the number `x` corresponding to the player.

- `tx_hero_id`: Integer. ID of the hero chosen by player `x` of team `t`. Categorical Feature.
- `tx_kills`: Integer. The number of enemy heroes killed by the player. Numerical Feature.
- `tx_deaths`: Integer. The number of times the player has been killed. Numerical Feature.
- `tx_assist`: Integer. Number of times the player helped kill an enemy hero, but did not actually kill the enemy hero. Numerical Feature.
- `tx_denies`: Integer. The number of denials performed by the player. Denying is the act of last hitting a friendly unit, preventing enemy heroes from last hitting a friendly unit. Enemies do not receive gold from a denied unit. Numerical Feature.
- `tx_gold`: Integer. The total amount of gold that the player has collected over the course of the match. Numerical Feature.
- `tx_lh`: Integer. The number of the player's last hits. If a player scores a last hit on an enemy or neutral NPC, they will receive the corresponding gold. Numerical Feature.
- `tx_xp`: Integer. Experience points earned by the player during the match. Numerical Feature.
- `tx_health`: Integer. *Current* health of the player. Numerical Feature.

- `tx_max_health`: Integer. Maximum health of the player. Numerical Feature.
- `tx_max_mana`: Integer. A player's maximum amount of mana, which is generally used to perform abilities. Numerical Feature.
- `tx_level`: Integer. The current level of the player. By gaining enough experience points, the player levels up, gaining more power and increasing their stats. Levels range from 1 to 25. Numerical Feature.
- `tx_x`: Integer. The player's X coordinate on the map. Numerical Feature.
- `tx_y`: Integer. The player's Y coordinate on the map. Numerical Feature.
- `tx_stuns`: Float. The total amount of time the player is stunned. Stun is a condition that prevents a player from taking almost any action. Numerical Feature.
- `tx_camps_stacked`: Integer. Number of camps stacked by the player. Camp stacking is the process of pulling neutral creeps(NPCs) away from their spawn points, known as camps. By pulling neutral creeps out of their camps, the game spawns a new set of creeps for the player to interact with, generating additional rewards and speeding up the farming process. Numeric Feature.
- `tx_creeps_stacked`: Integer. Total number of creeps taunted by the player during camp stacking. Numeric Feature.
- `tx_rune_pickups`: Integer. Number of runes collected by the player. A rune gives a bonus to all players in the team that collects it. Numerical Feature.
- `tx_firstblood_claimed`: Boolean Integer (0 or 1). Indicates whether the player made the first kill in the game. Categorical Feature.
- `tx_teamfight_participation`: Float. The ratio of a player's participation in fights with more than 2 players. Numerical Feature.
- `tx_towers_killed`: Integer. The number of towers destroyed by the player. Towers are the first line of defence in an enemy base, so destroying them is a milestone in the match. Numerical Feature.
- `tx_roshans_killed`: Integer. The number of Roshans the player has killed. Roshans are the most powerful neutral NPCs on the map, and killing them results in a huge reward for the team that kills them. Numerical Feature.
- `tx_obs_placed`: Integer. Number of observers placed by the player. The map is covered by *fog of war* that obscures the zone, the observer is an invisible object that clears the *fog* in the area for the team that placed it. Numerical Feature.
- `tx_sec_placed`: Integer. Number of sentries placed by the player. The sentry is an invisible object that gives the player's team the ability to see any invisible object or entity within the radius of the sentry. Enemy observers within sight of the sentry are blocked. Numerical Feature.

There are 24 single player features for each of the 5 players on the 2 teams, for a total of 240 player features.

2.2 Methodology

This project was written in Python with the help of the following libraries: sklearn, numpy, matplotlib, seaborn and pandas.

The dataset belongs to a competition hosted on Kaggle[1] and it is stored in CSV files.

The objective of the competition is to produce results with a model of the given test CSV, the results are evaluated using ROC-AUC score.

Along the train and test set, raw data stored in JSON is provided, allowing for further feature extraction.

A wide range of methodologies are used in this project:

- **Plot of the Dataset**: using tables and graphs thanks to the libraries mentioned.
- **Feature Extraction**: agglomeration of numerical features and One Hot Encoding of categorical features.
- **Feature Selection**: Intrinsic FS using Random Forest, Sequential FS, PCA and Kernel PCA for dimensionality reduction.
- **Classifiers**: Random Forest, Ada-boost, Gradient Boost and SVM.
- **Training**: K-fold cross-validation with stratification, exhaustive grid search for fine tuning.
- **Metrics**: ROC-AUC, accuracy, precision, recall and f1-score.

2.3 Experimentation Process

2.3.1 Information about Dataset

The JSON file is likely to have been generated directly from Dota 2 API calls, and thus it contains extensive information about each game and each player. The CSV files provided by the competition provide a reasonable level of information without retaining the most complex details, in this project no additional features from the JSON were added to the CSV dataset.

In this dataset, in addition to the standard features used to describe the state of the game such as `r1_kills` or `d3_x`, there are some non-immediate features : `lobby_type` and `game_mode`. To understand the values of these, dota2api responses[5] were consulted. In the dataset the only values of `lobby_type` present are 0 and 7, corresponding to public match and ranked match. Instead, `game_mode` has a wider range of values, the only ones of interest are 2 and 22, which represent the standard game modes for Dota 2 (Captain's Mode and Ranked All Pick).

2.3.2 Preprocessing

The feature `lobby_type` is dropped, and since it already contains the only relevant values, no row filtering is required.

The dataset keeps all rows that have the feature `game_mode` with a value of 2 or 22. In this way, only standard game modes are considered and not some alternative ones that are usually created for fun and not for competition. The dataset filters a total of 7502 rows. After filtering the rows, this label is dropped as deemed not relevant for predictions.

The feature `game_time` represents the time in the game at which the information needed to make the prediction is presented. After checking the data, some lines with `game_time=0` have already destroyed targets (e.g. towers) and killed enemies, so it was decided to drop all 20 lines with `game_time=0`, as these are probably the result of replacing NAN with 0 in `game_time` from the APIs.

Each player has `tx_teamfight_participation` feature that represents the percentage of teamfights the player has participated in. A few rows appear with a value >1 for at least one player, with a maximum of 1.2. It was decided to simply replace these values with 1.

The features `chat_len` and `match_id_hash` are dropped, as they are not considered relevant for predictions.

The preprocessing phase changes the shape of our dataset from (39675, 246) to (32153, 242). It is important to note that each player has a categorical feature to deal with, which is `tx_hero_id`. There are a total of 115 different heroes in total. These features will be handled by one-hot encoding with 3 different approaches in the next phase.

2.3.3 Decision of dataset

As mentioned above, one-hot encoding is implemented to handle the `tx_hero_id` feature, 3 different datasets are created:

- **Playerstats_playerheroes**: considering the stats for each player (features already present in the dataset) and the hero they use, this leads to the addition of 1,140 columns via one hot encoding.
- **Playerstats_teamheroes**: take into account the stats for each player (already in the dataset), but not the heroes of each player, only the presence of a hero in a team (radiant or dire) is taken into account with the new features `r_{heroid}` and `d_{heroid}`. This way, one-hot encoding only adds 220 columns as final result, since all the single-player hero features are dropped (`tx_hero_id`).

- **Teamstats_teamheroes**: considering the presence of a hero in a team and not single-player heroes, as well as not considering individual stats for each player: generalised team stats features as `t_gold` are introduced as the sum of all player stats of that team (`tx_gold`). The only features representing individual player information that are not modified are the positions on the map as `tx_y`. This approach results in the addition of 52 features in total as all individual stats features are dropped in the transformation (except for positions).

A **Random Forest** classifier with max_depth set to 10 is applied to these datasets in order to determine the importance of the features and to determine the best dataset one to use.

Dataset	Accuracy	Roc Auc
Playerstats_playerheros	0.696781	0.688976
Playerstats_teamheros	0.696781	0.687344
Teamstats_teamheros	0.704712	0.696491

Table 1: Accuracy and Roc Auc score of initial dataset variations

Dataset	Most important features
Playerstats_playerheros	r2_y, r4_y, r5_x, d3_x, d1_x, r5_y, r1_y, r1_x, d2_x, d3_y
Playerstats_teamheroes	r3_x, r1_y, d3_y, r3_y, r2_x, r4_y, d5_x, d3_x, r5_y, r1_x
Teamstats_teamheroes	r_deaths, r_towers_killed, d_towers_killed, d_kills, d_deaths, r_kills, r1_y, d_gold, d2_y, r_gold

Table 2: 10 most important features of initial dataset variations, first is the most important

Teamstats_teamheroes seems to be the most promising, the model even considers some team stats as the most important features (`towers_killed, deaths, kills and gold`).

Apart from some team stats, all classifiers agree that the position of the players is really important, in order to improve this aspect, features representing an overall position of the team are considered instead of individual ones, to do this 2 additional datasets are introduced and tested:

- **Team_mean_position**: add the features that represent the arithmetic mean of the positions of all players in the same team (`radiant_avg_x, radiant_avg_y, dire_avg_x, dire_avg_y`) and dropping all individual player positions, thus reducing the number of features by 16.

- **Team_weighted_mean_position**: add features by considering a weighted arithmetic mean of the position of all players in the same team, each position has a weight w calculated in this way:

$$w = \frac{1}{mean\ distance\ between\ all\ other\ teammates}$$

This way the team's position will be more centred around the more densely populated area (`radiant_Weighted_avg_x`, `radiant_Weighted_avg_y`, `dire_Weighted_avg_x`, `dire_Weighted_avg_y`). Then remove all individual player positions, reducing the number of features by 16.

Dataset	Accouracy	Roc Auc
Team_mean_position	0.714197	0.709707
Team_weighted_mean_position	0.719484	0.713676

Table 3: Accuracy and Roc Auc score of dataset variations with mean and weighted mean

Dataset	Most important features
Team_mean_position	`radiant_avg_y`, `dire_avg_x`, `dire_avg_y`, `radiant_avg_x`, `d_towers_killed`, `r_kills`, `r_deaths`, `d_kills`, `d_deaths`, `r_towers_killed`
Team_weighted_mean_position	`dire_Weighted_avg_y`, `radiant_Weighted_avg_y`, `radiant_Weighted_avg_x`, `dire_Weighted_avg_x`, `d_towers_killed`, `d_kills`, `r_towers_killed`, `d_deaths`, `r_deaths`, `r_kills`

Table 4: 10 most important features of variations with mean and weighted mean, first is the most important

As both accuracy and ROC-AUC improve and the added features are considered very important, the project will consider the **Teamstats_teamheroes** dataset and its two variants **Team_mean_position**, **Team_weighted_mean_position**.

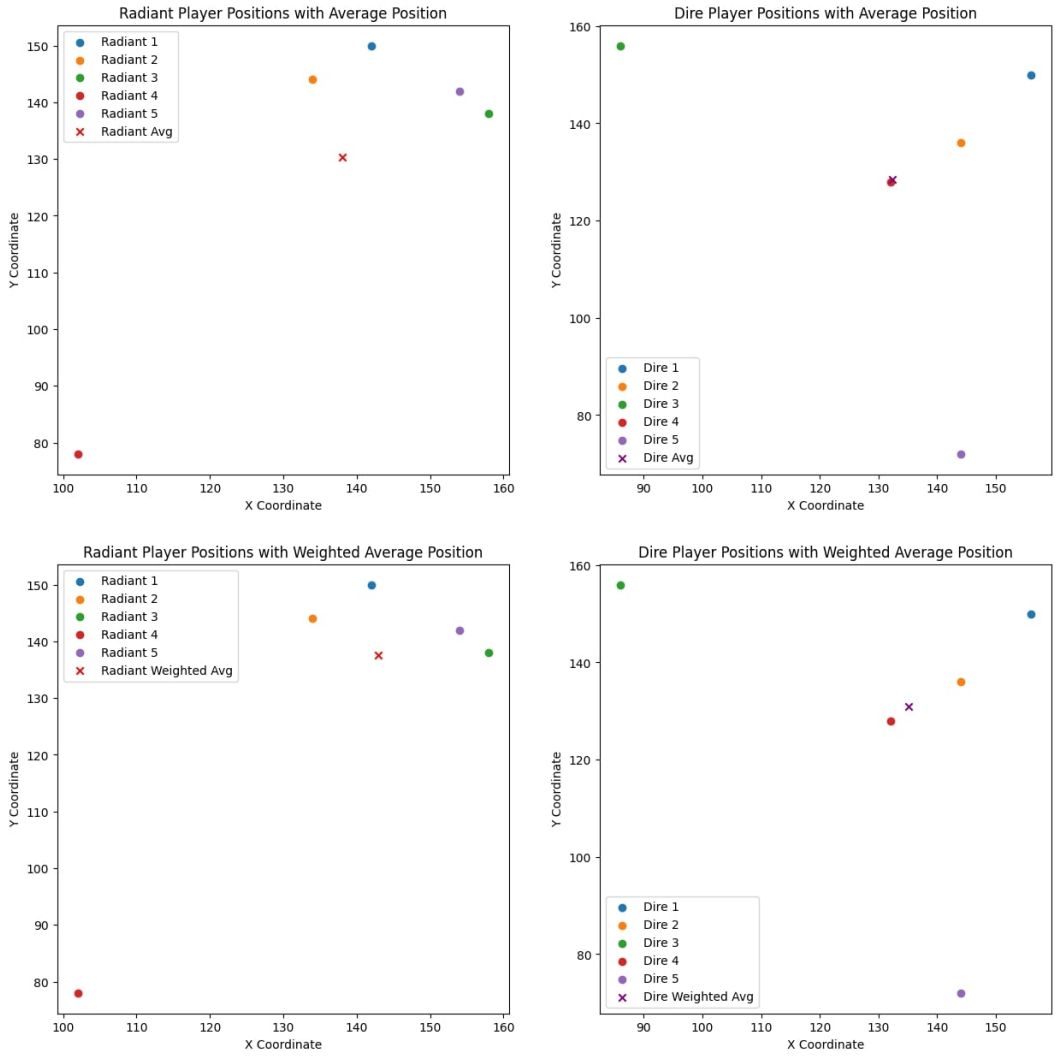


Figure 2: Plot comparison between mean and weighted mean positions

All the procedures done so far in the section 2.3 are available in the notebook **feature_importance.ipynb**.

2.3.4 Feature Selection

Due to the large number of features in all the datasets created, more methods of feature selection have been introduced.

The first method is a **Intrinsic Feature Selection IFS** using **Random Forest**, the model measures the importance of the features as described in subsection 2.3.3 and has a `max_depth` set to 10 to speed up the process and retrieve the **information** from the trees at the highest levels. All features with an importance below a fixed **threshold** of 0.01 were discarded, thus reducing the number of features.

Retraining the model that computed the importance with the filtered dataset shows an **improvement** in the metrics, the result can be seen in table 5.

Another method tested was the **Forward Sequential Feature Selection FSFS** along with Random Forest with the number of features selected not specified and a **tolerance** of 0.01. The FSFS adds and replaces features until the performance is no longer improved by the tolerance.

Only 4 features were selected with the FSFS, but the metrics obtained from this reduced dataset are far below those reduced by Intrinsic, one hypothesis being that the information contained in the four columns is too low. Another reason for rejecting this feature selection approach was the long computation time, around 30 minutes.

Finally, two dimensionality reduction methods were tried for feature visualisation and selection: linear **PCA** and non-linear **Kernel PCA**.

The idea is to reduce the presence of correlations between features, as shown in figure 3.

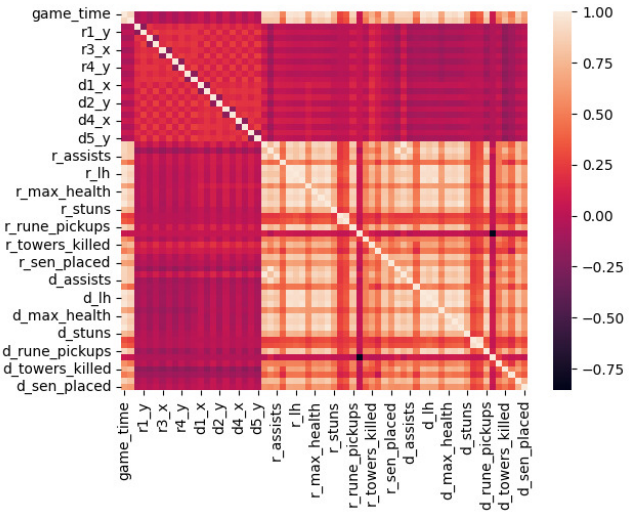
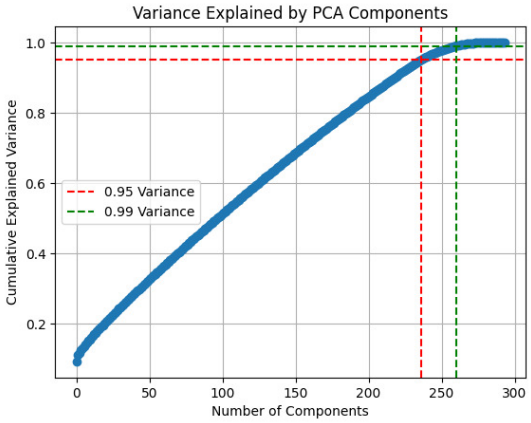


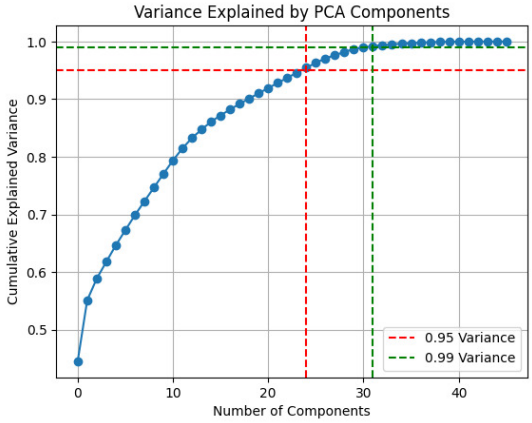
Figure 3: Pearson's Correlation between features without heroes features

PCA is very fast and can optimally reduce the dimensions while preserving the information when the thresholds of explainable variance are set at 0.95 and 0.99 for the datasets with an IFS already applied; the IFS removes a lot of irrelevant features generated by the one-hot encoding

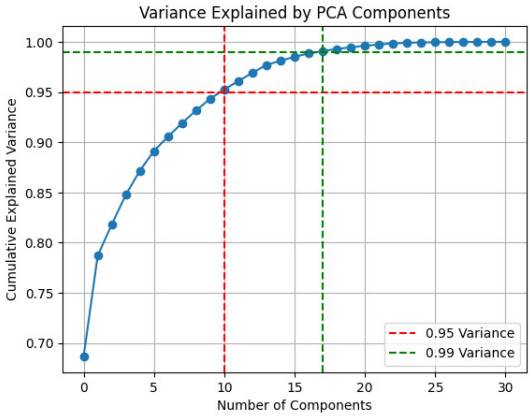
of the categorical features and allows PCA to reduce the correlation.



(a) **Teamstats_teamheros**



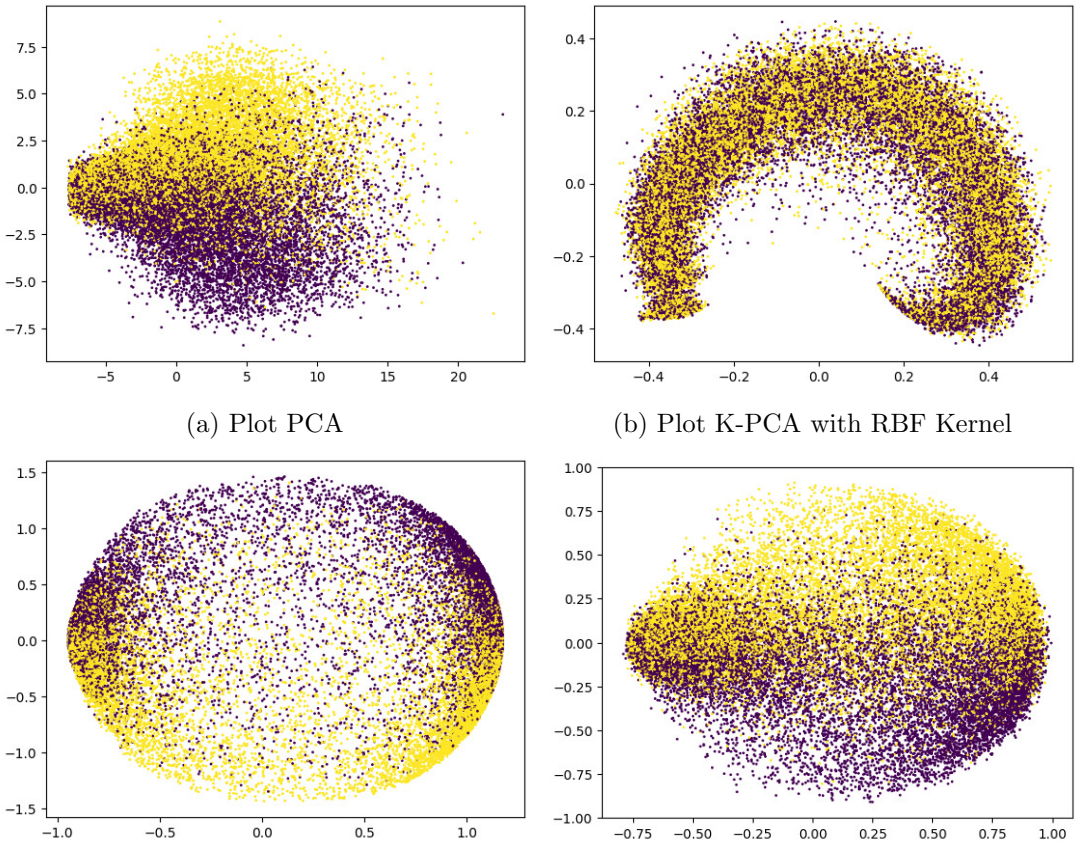
(b) **Teamstats_teamheros** after Intrinsic FS



(c) **Team_mean_position** after Intrinsic FS

Figure 4: Comparison of Explainable Variance ratio with different dataset

Kernel PCA is very slow, this is due to computing the kernel trick for all samples and then performing the PCA. It has been used the *randomize* version, for retrieving the eigenvectors, this has accelerated the process, but still very slow compared to PCA, also the 2D plot does not seem to create separability on the data in a way to help model such as SVM or K-nearest neighbours, for these reasons this method has been discarded.



(a) Plot PCA

(b) Plot K-PCA with RBF Kernel

(c) Plot K-PCA with Sigmoid Kernel

(d) Plot K-PCA with Cosine Kernel

Figure 5: Comparison with different type of Kernel PCA

In the end, only the **Intrinsic FS** and the **PCA** were used for some of the models, with particular emphasis on the Intrinsic one, the aim is still to have an explanation of the prediction through the features, while PCA lacks **interpretability** but has a great dimensional reduction while preserving the information in the eigenvalues, so we reserved this method for the slow models that suffer from the curse of dimensionality.

The table 5 shows the difference between the different datasets **Teamstats_teamheroes** (**TT**), **Team_mean_position** (**TMP**) and **Team_weighted_mean_position** (**TWMP**); all tests were run using the same **RandomForest** model along a `train_test_split` with a validation size of 0.2.

Dataset	Old Feat.	New Feat.	Old Roc-Auc	New Roc-Auc
TT with IFS	294	46	0.710653	0.719138
TMP with IFS	278	31	0.717262	0.721185
TWMP with IFS	278	32	0.711152	0.711714
TT with FSFS	294	4	0.710653	0.682918
TT with IFS and PCA at 0.99	294	32	0.710653	0.712666
TT with IFS and PCA at 0.95	294	25	0.712666	0.720941
TMP with IFS and PCA at 0.99	278	18	0.717262	0.715997
TMP with IFS and PCA at 0.95	278	11	0.717262	0.709935
TWMP with IFS and PCA at 0.99	278	18	0.711152	0.718376
TWMP with IFS and PCA at 0.95	278	11	0.711152	0.715440

Table 5: Comparison between Feature Selection Methods

All the procedures done in section 2.3.4 are present in the notebook **feature_selection.ipynb**.

2.4 Models tested

After the feature extraction process, the following models are trained on **Teamstats_teamheroes** and its two variants with team mean position and team weighted mean position.

Although the size of the dataset is large, it was decided to perform a **Stratified K-fold Cross Validation** in order to achieve consistency in the choice of hyperparameters rather than speed. When the model was fast, an exhaustive search of the hyperparameters was performed in combination with the cross-validation using GridSeachCV. For heavier model, the train was implemented on Google Colab platfrom.

- **Random Forest**: trained using GridSearchCV with StratifiedKFold on 5 splits, the parameters tested for the model are
  - `n_estimators`: [50, 100, 150]
  - `class_weight`: ['balanced', `None`]
  - `criterion`: ['entropy', 'gini']
  - `max_depth`: [5, 10]
  - `min_samples_leaf`: [1, 2, 5]

  Even though the dataset targets are very balanced, it was decided to include a `class_weight` parameter to see if a small weight towards the 'smallest' class could affect the ROC-AUC, while `max_depth` and `min_samples_leaf` were included to increase the generalization of the forest, as this type of model has a high risk of overfitting.

- **K-nearest neighbours**: trained using GridSearchCV with StratifiedKFold on 5 splits. All dataframes were first reduced with PCA at 0.95 and 0.99 variance and then standardised with MinMaxScaler before training. In total, there were 6 different data frames. The tested parameters of the model are

  - `n_neighbours`: all odd numbers from 1 to the square root of the number of samples

  This type of model does not scale well with high-dimensional feature data, which is why PCA was introduced. As this type of model is very sensitive to scale, it was decided to standardize it before training.

  Even after taking these transformations into account, the model prediction was still very slow, as a compromise only `n_neighbours` was included as a hyperparameter and the two smallest datasets were tested.

- **AdaBoost**: trained using GridSearchCV with StratifiedKFold on 5 splits. The estimator used is a **DecisionTreeClassifier**. The parameters tested for the model are

  - `estimator__max_depth`: [1, 2, 3]
  - `n_estimators`: [50, 75, 100]
  - `learning_rate`: [1, 1.5, 2]

  Again, `estimator__max_depth` has been inserted to keep the overfitting of the model in check, while AdaBoost seems to prefer higher instances of `learning_rate` for this task, contrary to common usage.

- **Histogram-Based Gradient Boosting**: trained using GridSearchCV with StratifiedKFold on 5 splits. HistGradientBoosting was implemented because it is much faster than Gradient Boosting for large datasets (n_samples $\geq$ 10 000). The tested hyperparameters for the model are:

  - `l2_regularization`: [0, 0.1, 0.01]
  - `max_features`: [0.5, 0.75, 1.0]
  - `min_samples_leaf`: [50, 60, 70]
  - `max_iter` = [200]
  - `validation_fraction` = [0.05]

  To increase the generalization of this model the parameters `l2_regularization`, `max_features` and `min_samples_leaf` have been introduced.

  While the number of iterations has been set higher than the recommended values, by default there is a **early-stopping** mechanism that halts training if the score on the validation fraction has not improved. The `validation_fraction` was kept low because the data set was already reduced by K-folding.

- **SVM**: trained using HalvingGridSearchCV with StratifiedKFold on 5 splits. The tested hyperparameters of the model are
  - `kernel`: ["rbf"]
  - `C`: $[2^{-4}, 2^{-2}, 2^{-1}]$
  - `gamma`: $[2^{-3}, 2^{-2}, 2^{-1}]$

  As SVM is a very slow model for large datasets, but is independent from the size of the target space, the PCA transformation was not performed and HalvingGridSearchCV was used.

  This object divides the dataset into small random subsets and performs training for all combinations of hyperparameters and selects only the best only the best, then re-performs training with larger subsets until there is only one set with all training data and a pair of hyperparameters to compare.

  HalvingGridSearchCV is significantly faster than GridSearchCV, the only downside is that only one scoring metric can be displayed at a time, `roc_auc` is selected.

3 Results

3.1 Random Forest

Dataset	roc_auc	accuracy	recall	precision	f1
Teamstats_teamheroes	0.799801	0.715610	0.762583	0.716364	0.738697
Team_mean_position	0.803675	0.718471	0.759101	0.721444	0.739749
Team_weighted_mean_position	0.803117	0.716760	0.812024	0.699187	0.751374

Table 6: Comparison of Test Scores across Datasets (Random Forest)

Dataset	class_weight	criterion	max_depth	min_samples_leaf	n_estimators
TT	balanced	gini	10	5	150
TMP	balanced	entropy	10	5	150
TWMP	None	entropy	10	5	150

Table 7: Best Hyperparameters for each Dataset (Random Forest)

3.2 K-nearest Neighbours

Dataset	roc_auc	accuracy	recall	precision	f1
TMP PCA 0.95	0.788106	0.704413	0.789781	0.692647	0.738003
TMP PCA 0.99	0.789683	0.704164	0.792967	0.691263	0.738622
TWMP PCA 0.95	0.787346	0.704164	0.798041	0.689615	0.739869
TWMP PCA 0.99	0.787250	0.704476	0.798100	0.689929	0.740070

Table 8: Comparison of Test Scores across Datasets (K-nearest neighbours)

Dataset	n_neighbours
Team_mean_position PCA 0.95	169
Team_mean_position PCA 0.99	163
Team_weighted_mean_position PCA 0.95	179
Team_weighted_mean_position PCA 0.99	163

Table 9: Best Hyperparameters for each Dataset (K-nearest neighbors)

3.3 AdaBoost

Dataset	roc_auc	accuracy	recall	precision	f1
Teamstats_teamheroes	0.789926	0.707585	0.752964	0.709934	0.730784
Team_mean_position	0.795377	0.710074	0.756859	0.711670	0.733423
Team_weighted_mean_position	0.793450	0.710478	0.762465	0.709853	0.735189

Table 10: Comparison of Test Scores across Datasets (Adaboost)

Dataset	estimator_max_depth	learning_rate	n_estimators
TT	2	1.5	100
MP	2	1.5	100
WMP	2	1	100

Table 11: Best Hyperparameters for each Dataset (Adaboost)

3.4 Histogram-Based Gradient Boosting

Dataset	roc_auc	accuracy	recall	precision	f1
Teamstats_teamheroes	0.806958	0.719995	0.770488	0.718683	0.743661
Team_mean_position	0.809771	0.721799	0.765118	0.723235	0.743547
Team_weighted_mean_position	0.810007	0.722763	0.770134	0.722336	0.745465

Table 12: Comparison of Test Scores across Datasets (Histogram-Based Gradient Boosting)

Dataset	l2_regularization	max_features	min_samples_leaf
TT	0.1	0.75	60
MP	0.1	0.5	70
WMP	0.1	0.5	50

Table 13: Best Hyperparameters for each Dataset (Histogram-Based Gradient Boosting)

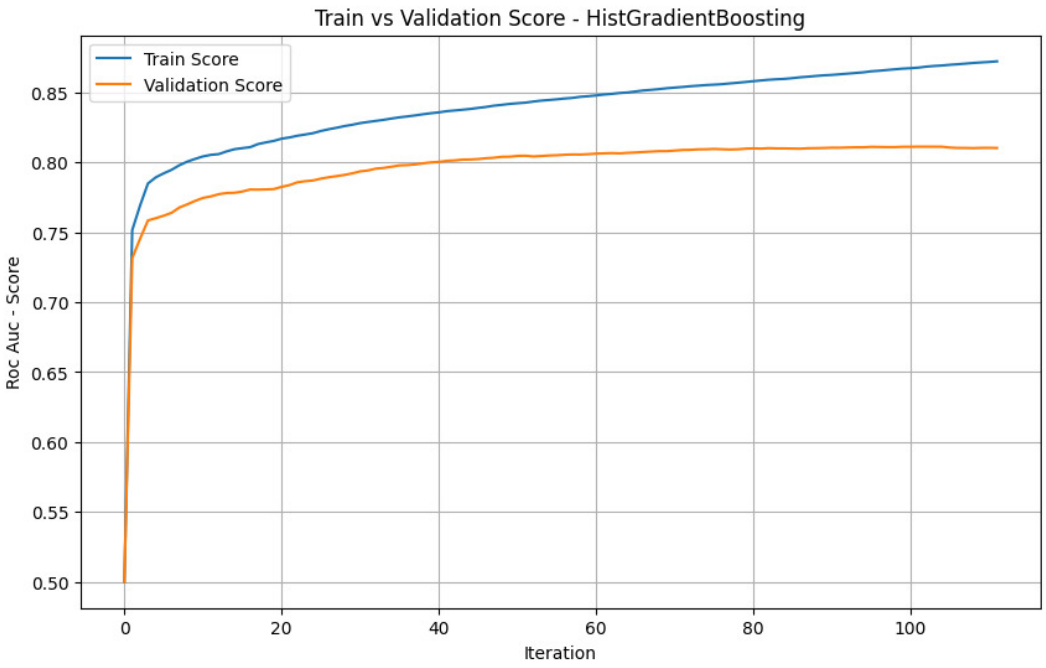


Figure 6: Comparing ROC-AUC for Train and Valdation over iterations

3.5 Support Vector Machines

Dataset	roc_auc
Teamstats_teamheroes	0.769465
Team_mean_position	0.798335
Team_weighted_mean_position	0.798220

Table 14: Comparison of Test Scores across Datasets (SVM)

Dataset	C	Gamma	Kernel
TT	0.5	0.125	rbf
MP	0.5	0.125	rbf
WMP	0.5	0.125	rbf

Table 15: Best Hyperparameters for each Dataset (SVM)

4 Conclusion

The objective of this project was to build a classifier capable of predicting the winning team in a real-time Dota 2 match with a ROC-AUC score above 0.8. To achieve this, after pre-processing the dataset, different feature extraction approaches were applied, resulting in a dataset with summarising team features in terms of individual player features and its two variants, each incorporating a mean or median position of the teams.

Different feature selection approaches were tried: **Intrinsic Feature Selection**, **Sequential Feature Selection** and **PCA**.

The **Intrinsic Feature Selection** proved to be the key to good feature selection, prioritising the explainability of the feature to see what is more relevant to winning a Dota 2 match.

The tested models were **Random Forest**, **K-Nearest Neighbours**, **AdaBoost**, **Histogram-Based Gradient Boosting** and **Support Vector Machines**.

Model	Dataset	ROC-AUC	F1 Score
HistGradientBoost	WMP	0.810007	0.745465
RandomForest	TMP	0.803675	0.739749
SVC	TMP	0.798335	0.751812
AdaBoost	TMP	0.795377	0.733423
k-NN	TMP PCA 0.99	0.789683	0.738622

Table 16: Comparison of Test Scores across Models and Datasets with corresponding best Hyperparameters

Table 16 shows that the best performing model is the Gradient Boosting model, with a ROC-AUC of 0.810007. It was the one chosen for the competition, resulting in a Kaggle score of 0.81935. The state-of-the-art approach of the competitions achieved a Kaggle score of 0.85909[2] and 0.81293[3].

The best **ensemble methods** also agree that the most important feature is the average position, while the least relevant is surprisingly the game time. This confirms that merging positions was a good choice.

In conclusion, all objectives have been met, but there are two improvements that can be made:

- Trying a new boosting model or focusing on gradient boost, increasing the range of hyperparameters, using GPU versions to speed up the process.
- Trying to data mine the JSON file and retrieve more features that can be worked with to try to go beyond the 0.81 limit of the ROC-AUC score.

References

[1] Peter Romov and Yury Kashnitsky. *mlcourse.ai: Dota 2 Winner Prediction*. `https://kaggle.com/competitions/mlcourse-dota2-win-prediction`. Kaggle. 2019.

[2] Andrei Shchahlou. *DOTA 2 with time*. `https://www.kaggle.com/code/andapka/dota-2-with-time`. Kaggle. 2020.

[3] Denis Mikheev. *Combine hero features into team ones - basic*. `https://www.kaggle.com/code/daemonis/combine-hero-features-into-team-ones-basic`. Kaggle. 2019.

[4] Valerii Chetvertakov. *Dota 2: engineering coordinates features*. `https://www.kaggle.com/code/chetvertakov/dota-2-engineering-coordinates-features`. Kaggle. 2020.

[5] *dota2api responses*. `https://demodota2api.readthedocs.io/en/latest/responses.html`.