**University of Pisa**

**Data Mining and Machine Learning**

****

**League of Legends match predictor**

**-**

**Design and development of an application able to predict a team’s victory in a League of Legends game using data mining techniques and algorithms**

**Davide Vigna**

**Summary**

[**Description** 2](#_Toc75984274)

[**KDD Process** 3](#_Toc75984275)

[Dataset building 4](#_Toc75984276)

[First Dataset 4](#_Toc75984277)

[Second Dataset 4](#_Toc75984278)

[Integrated Raw Dataset 5](#_Toc75984279)

[Data pre-processing 7](#_Toc75984280)

[Feature Construction 8](#_Toc75984281)

[Feature Extraction 8](#_Toc75984282)

[Feature Transformation 9](#_Toc75984283)

[Feature Reduction 9](#_Toc75984284)

[Validation 11](#_Toc75984285)

[Filtering 11](#_Toc75984286)

[Data Mining 12](#_Toc75984287)

[Random Forest 12](#_Toc75984288)

[K-Nearest Neighbour 13](#_Toc75984289)

[Naïve Bayesian 13](#_Toc75984290)

[Support Vector Machine (SVM) 14](#_Toc75984291)

[Evaluations of results 15](#_Toc75984292)

[Observations 20](#_Toc75984293)

[Conclusions 21](#_Toc75984294)

[**Application** 22](#_Toc75984295)

[Analysis Phase 22](#_Toc75984296)

[Design Phase 23](#_Toc75984297)

[Implementation phase 25](#_Toc75984298)

[Test phase 28](#_Toc75984299)

[**User manual** 29](#_Toc75984300)

[Preliminary Actions 29](#_Toc75984301)

[How to use the application 29](#_Toc75984302)

# **Description**

**What is League of Legends?**

**League of Legends** is a team-based strategy online game where two teams of five players face off to destroy the other’s core base called ‘Nexus’. Each player controls a champion that has specific skills and it is chosen in the pre-match phase.

The game map is the same for all ranked games. Each team has its own base, which contains the spawn and respawn point (where the champions start the game and where "reborn" after dying), the Nexus, inhibitors (particular Nexus defence structures) and some towers (generic defence structures that attack enemies within their firing range). The two bases are connected through three ways called lanes (top, mid and bottom). At regular intervals, waves of NPCs called ‘minions’ appear from the Nexus and autonomously advance through the lanes, fighting any enemies they encounter. In the space between lines, there is an area called jungle where neutral monsters live. These monsters can be killed by a champion for gold and experience bonuses, like for the “minions” in the lanes. Some neutral monsters give their killers a temporary boost (or buff) that will help them in battle (like Heralds, Dragons). In the lanes and in the jungle, there are also tall grass bushes that make the champions who enter inside invisible, useful to run away from enemies and not to be noticed and to ambush opponents. Champions can put a limited number of wards inside them, in order to have better visibility of the game.

During the game, each player must earn experience and gold by killing opposing champions and minions and by defeating neutral monsters. With the gold earned the player can buy items to equip his/her champion in order to improve own advantage. More experience a champion gains, higher his/her level and his/her skills will be.

Like all strategy games, there is no specific strategy to win. A typical League of Legends game duration can vary from 25 to 45 minutes. Each game can be divided into two phases: the *laining phase* and the *fight phase*. Players normally spend the first 10 to 15 minutes to gain early advantages in builds and levels in their own lane/area. In the remaining time, players start to focus on the macro level: take down towers, get map objectives and group fights.

Ranked games are types of matches in which each player is assigned skill rating, with the aim of creating balanced teams. The scale goes to the lower degree *Iron* to the maximum degree *Challenger*.

**Aim of this project**

The purpose of this work is to analyze the match information during the *laning phase* and understanding which of the overall characteristics contribute to lead a team to victory. In the future, this project could be integrated into the real League of Legends game with the aim of providing to the two teams the information of who has the best chance of winning during a match.

# **KDD Process**

## Dataset building

The dataset used in this project is made up of two different sources with the goal of building a model that takes in consideration matches from the rank *Gold* *(Medium*) to the rank *Challenger (*High*)*. All matches with rank lower than *Gold* are not considered because lot of them are composed by people with low experience and their game behaviour could influence the final result of the data mining process.

Both the sources are obtained with a scraping technique using some Python scripts. They make requests to the API *Riot Developer Portal* [[1]](https://developer.riotgames.com/) (authentication is required). Once logged, to retrieve match information it’s mandatory to have a match identification. The Riot web services return a response in JSON format. This response is elaborated in order to retrieve match information desired.

The next part explains how list of match ids has been retried for both the sources.

### First Dataset

The first source contains match information about rank Gold.

A list of match ids is obtained executing the *Extract\_MatchIdGold.py*. Starting from a *Gold* user account, the script discovers the list of the last 100 matches associated to the account and for each one, it tries to find all the account players participated in that game. The same procedure is repeated for each discovered account several times until a stop condition (achievement of 26299 matches). To avoid duplicating information, if a discovered match id is already in the list, the match is discarded.

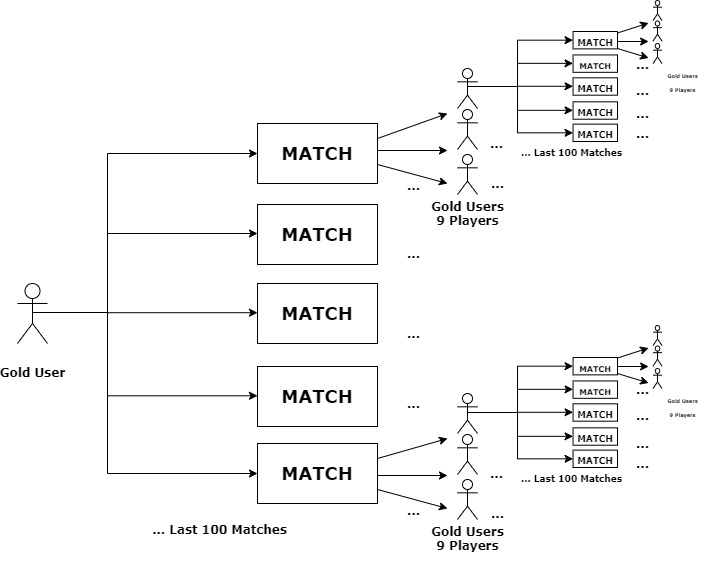


Figure 1 Procedure for retrieving match ID by scraping

### Second Dataset

The second source contains match information about high ranks (Challenger, Grandmaster, Master). This dataset is obtained downloading a well populated one at link [[2]](https://www.kaggle.com/fernandorubiogarcia/league-of-legends-high-elo-patch-1016) but filtering its content and considering just the list of match ids. This choice has been done for several reasons:

1. Mix information: some tuples refer to different kind of matches, different from “ranked”.
2. Not Complete: some tuples refer to matches with game duration lower than 15 min and there is no information about the match at minute 10 and 15.

The *Extract\_MatchIdChallenger.py* script generate a list of 25286 match ids.

The lists of match identifiers are given as input to the Python script *Extract\_RawDataset.py* that finally produces a csv file containing match information about the two team at minute 10 and 15.

The data integration procedure is quite easy because each dataset has the same attribute and there is no entity identification problem.

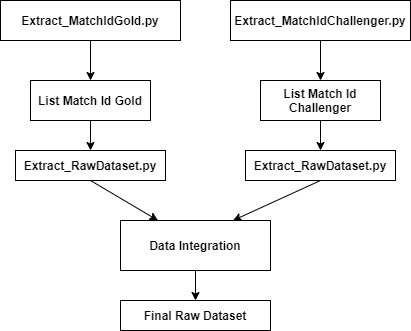


Figure 2 Block diagram for data integration

### Integrated Raw Dataset

The final row dataset is composed by 51585 instances with 79 attributes. The first attribute is the match identification number: *gameId*. Each of the following 78 attributes represents a feature of a team (BLUE or RED) in a specific minute (10 or 15). In order to distinguish the team and the minute, it has been used the following convention: a generic column name is composed in the form like NOT\_MIN\_ATTR where NOT is the name of the team and can be Blue or Red, MIN identify the minute and can be 10Min or 15Min and ATTR is the attribute name.

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| Win | Binary | 1 if a team has won the game at the end |
| Current\_Gold | Numeric | the number of coins that a team can actually spend |
| Total\_Gold | Numeric | the number of coins that a team have earned so far |
| Total\_Level | Numeric | the sum of levels of each player in a team as far |
| TotMinionsKilled | Numeric | the number of minions killed from a team so far |
| TotJnglMinionsKilled | Numeric | the number of minions killed from a team so far, in the jungle area. |
| FirstKill | Binary | 1 if the team has done the first kill, 0 otherwise |
| Kills | Numeric | the total number of kills executing from a team so far |
| Deaths | Numeric | the total number of kills executing from a team so far |
| Assists | Numeric | the number of times a player in a team has participated in a kill |
| WardPlaced | Numeric | the total number of wards placed by a team so far |
| WardKilled | Numeric | the total number of enemy wards killed by a team so far |
| FirstTower | Binary | 1 if the team has destroyed the first tower, 0 otherwise |
| FirstInhibitor | Binary | 1 if the team has destroyed the first inhibitor, 0 otherwise |
| FirstTowerLane | Nominal | the line’s name in which the fist tower has been destroyed |
| TowerKills | Numeric | the total number of towers destroyed by the team so far |
| Inhibitors | Numeric | the total number of inhibitors destroyed by a team so far |
| FirstDragon | Binary | 1 if the team has killed the first dragon, 0 otherwise |
| Dragons | Numeric | the total number of dragons killed by the team so far |
| RiftHeralds | Numeric | the total number of rift heralds killed by the team so far |

The dataset is a bit imbalanced, so in order to have the same number of instances in which a team wins and loses some rows are removed from the dataset.

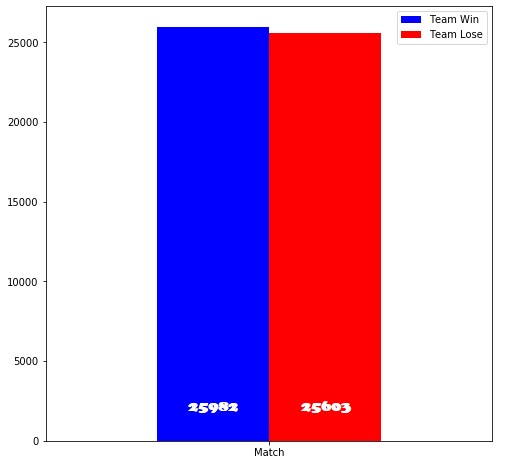


Figure 3 Comparison between two possible outcomes

## Data pre-processing

At this stage the data are transformed into forms appropriate for mining. A generic attribute in the raw dataset represents an information related to a team in a specific minute (10-15), so it is represented in an easier way in order to reduce the *course of dimensionality* problem with all the benefits that come from it.

The following table contains dataset information after the pre-processing phase:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| Team\_Win | Binary | 1 if BLUE team wins, 2 if RED team wins |
| DiffLevels | Numeric | the difference of levels of the two teams in (10-15 Min) |
| DiffMinions | Numeric | the difference of minions of the two teams in (10-15 Min) |
| DiffJglMinions | Numeric | the difference of jungle minions of the two teams in (10-15 Min) |
| DiffDragons | Numeric | the difference of dragons killed by the two teams in (10-15 Min) |
| DiffDomRatio | Numeric | the difference of the domination score of the two teams in (10-15 Min) |
| DiffTowers | Numeric | the difference of towers destroyed by the two teams in (10-15 Min) |
| DiffInhibitors | Numeric | the difference of inhibitors destroyed by the two teams in (10-15 Min) |
| DiffHeralds | Numeric | the difference of Heralds killed by the two teams in (10-15 Min) |
| DiffWardVision | Numeric | the difference of vision score of the two teams in (10-15 Min) |
| DiffTotGold | Numeric | the difference of gold earned by the two teams in (10-15 Min) |
| DiffTotGoldAvail | Numeric | the difference of gold available by the two teams in (10-15 Min) |
| FirstKill | Nominal | 1 if BLUE-2 if RED team commit the first kill |
| FirstTower10 | Nominal | 1 if BLUE-2 if RED team destroys the first tower at 10 min, 0 otherwise |
| FirstTower15 | Nominal | 1 if BLUE-2 if RED team destroys the first tower at 15 min, 0 otherwise |
| FirstInhibitor | Nominal | 1 if BLUE-2 if RED team destroys the first Inhibitor, 0 otherwise |
| FirstDragon10 | Nominal | 1 if BLUE-2 if RED team kills the first dragon at 10 min, 0 otherwise |
| FirstDragon15 | Nominal | 1 if BLUE-2 if RED team kills the first dragon at 15 min, 0 otherwise |
| BLUEFirstTowerLane | Nominal | the lane where team BLUE destroyed the first tower  (NONE, TOP, MID, BOT) |
| REDFirstTowerLane | Nominal | the lane where team RED destroyed the first tower  (NONE, TOP, MID, BOT) |

The next part contains all the pre-process technique adopted to obtain the pre-process dataset.

### Feature Construction

In the raw dataset there are some attributes that can’t been considered individually because they represent a particular concept in group. This involves the creation of new attributes that can capture the important information much more efficiently than the original attributes.

* *Domination Ratio*,represents how well a team is playing considering the attributes Kills, Deaths and Assists. It is calculated with the following formula:

This formula gives more relevance to the *deaths* attribute because lower the number of deaths is, better the team is playing.

* Ward Vision, represent how good is the team's vision in the map, taking in consideration the number of wards placed by a team and the number of wards destroyed by the opponent team. This value is calculated with the following formula:

Higher this number is, better the vision in the map is.

### Feature Extraction

Most of the attributes can be seen in terms of difference between attributes team at different minutes. In this way the information represented is the same and the number of attributes is less.

For example, the *DiffLevel* attribute has been computed applying this transformation:

The new introduced attribute replaces the other 4. This operation has been repeated for all the other numerical attribute.



Figure 4 Time frame Analysis (10-15Min)

### Feature Transformation

All binary attributes in raw dataset representing an event happened first, are considered just one per match(instance) instead of one per team. In the transformed dataset they represent the information of the team to which the event belongs. For example, the attribute *FirstInhibitor* is remapped as nominal attribute considering all the possible combination it can assume: 0,1,2 if respectively belongs to NONE, BLUE or RED team.

The same transformation is applied to all the other nominal attributes.

It’s necessary to use a *One Hot Encoding* technique in order to transform all the nominal attributes into numerical form. This method consists of introducing new variables on the number of unique values in the categorical feature. Using a different encoder, eg. Integer encoding, allows the model to assume a natural ordering between categories and it may result poor performance or giving unexpected results.

|  |  |
| --- | --- |
| Attribute | #Unique values |
| FirstKill | 2 |
| FirstTower10 | 3 |
| FirstTower15 | 3 |
| FirstInhibitor | 3 |
| FirstDragon10 | 3 |
| FirstDragon15 | 3 |
| BLUEFirstTowerLane | 4 |
| REDFirstTowerLane | 4 |

This choice was considered acceptable, despite introducing some new variables.

### Feature Reduction

This operation has been applied carefully in order to obtain a reduced representation of the dataset but yet produces the same analytical results.

The class is represented by the *Team\_Win* attribute. This information is redundant in the raw dataset because it is present a *Win* attribute for each team, so one of the two is removed and in the other one the values are remapped with a LabelEncoder.

The attribute *GameId* is considered as an irrelevant feature because it’s just an identification of a match and it can’t produce any benefits to the following mining phase.

The attribute *FirstKill*10 is considered redundant because its information is always the same present in attribute *FirstKill15*. The scenario where up to 10 minutes there are no kills committed and after that a kill is committed by a team is an outlier situation that in this dataset never appears.

The attribute *FirstInhibitor10* and *DiffInhibitor10* are considered redundant because their information are, in most of cases, the same present respectively in *FirstInhibitor15* and *DiffInhibitor15*.

A brief frequency analysis is performed on these two last attributes. In most of cases, *DiffInhibitor15* is equal to 0 because in 15 minutes it’s quite hard to destroy an inhibitor. However, there are some outliers in which this value is different from 0. This situation happens just once until minute 10 and 481 times until minute 15. After this consideration the attributes *FirstInhibitor10* and *DiffInhibitor10* have been removed.

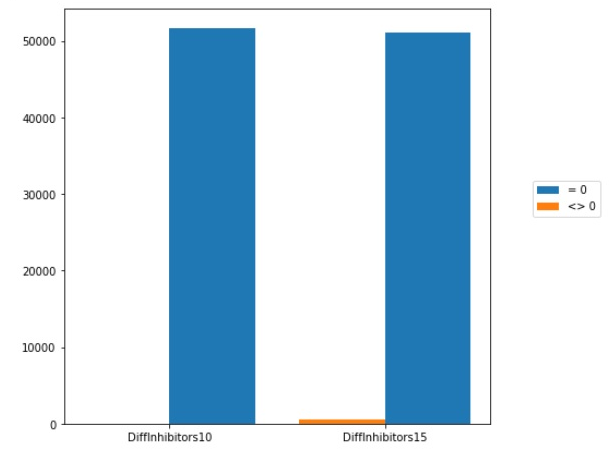
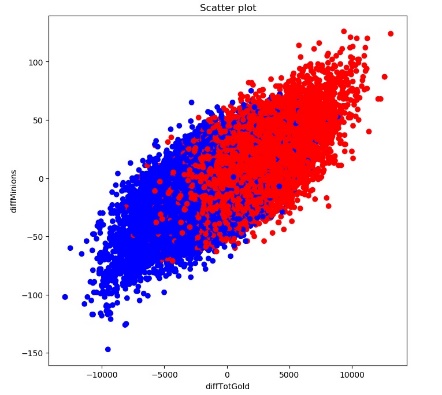
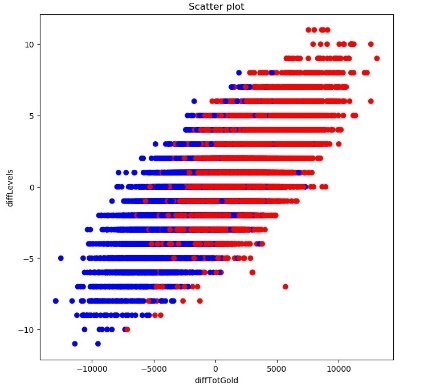
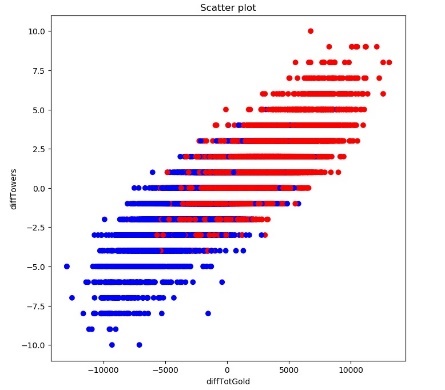


Figure 5 Compared histogram between DiffInhibitor at 10 and 15 minute

At the end of this phase the pre-processed dataset contains 37 attributes.

It’s interesting to observe the strong correlation between the attribute *DiffTotGold* and the attributes

*diffTowers, diffLevels* and *diffMinions*.

****

The *Pearson product moment coefficient* has been applied on each pair and on the class attribute obtaining the following results:

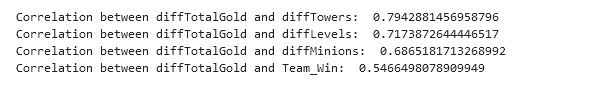


Figure 6 Results of correlation analysis

These correlations have sense because every time a player destroys a tower or level up or kills a minion, extra gold is rewarded. The *DiffTotGold* attributeis also the most correlated to the class attribute.

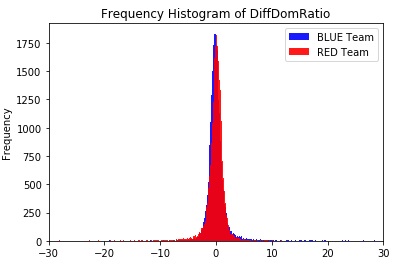
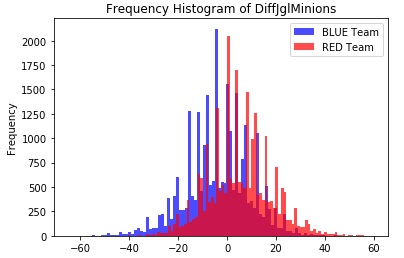
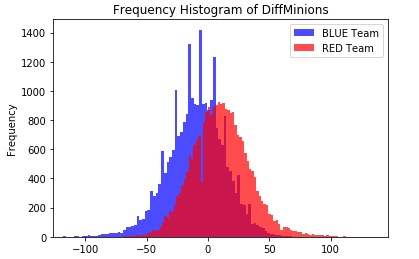
### Validation

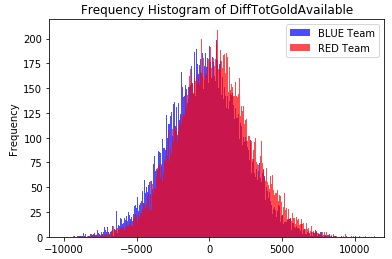
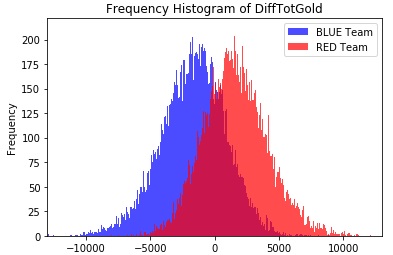
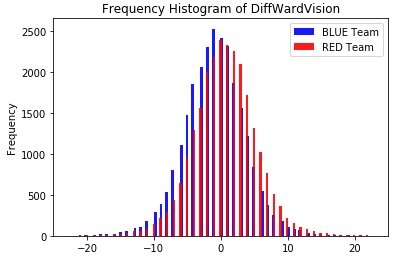
After the pre-processing phase, several tests have been done with different classifiers and for each performing a 10-Fold-Cross-Validation. This technique reduces a lot the phenomena of *model overfitting* because at each iteration it selects a different fold (train and test sets), despite it requires more execution time respect to the holdout validation method.

### Filtering

At each iteration of the 10-Fold-Cross-Validation the dataset is splitted into train set and test set and the following filters are applied on each train set fold in this order:

* Resampling: an under-sampling without replacement, in order to rebalance the class distribution during the building of the model [[3]](https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html);
* Discretization: the numeric attributes with most spread distribution are discretized using a supervised approach that replaces continuous numerical variables by discrete, finite, values estimated by a decision tree [[4]](https://feature-engine.readthedocs.io/en/1.0.x/discretisation/DecisionTreeDiscretiser.html) [[5]](https://towardsdatascience.com/an-introduction-to-discretization-in-data-science-55ef8c9775a2);
* Normalization: min-max normalization transforming the original data linearly, and ensure that the new data are mapped to the [0,1] interval, to achieve the proportional scaling of the original data [[6]](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html);
* Attribute Selection: selecting the first 10 best attributes according to the *chi-quare* score function [[7]](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html).





These images show the frequency distribution of the continues attributes with most spread on which the discretization filter has been applied.

## Data Mining

This aim of this project is to perform a binary classification and the class information is represented by the attribute *Team Win*:

* Team Win = 0 means, team blue wins
* Team Win = 1 means, team red wins

In order to create a model able to predicting classes of new instances, different types of classification algorithms are evaluated on the dataset and their results compared.

The metrices used for comparisons are *accuracy*, *precision*, *recall* and *f-measure*. The *time* needed for the model's training and *overfitting* are also considered.

### Random Forest

The Random Forest algorithm is an *ensemble* classification algorithm. It consists of a group of decision trees (forest). It works in four steps:

1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction. [[8]](https://www.datacamp.com/community/tutorials/random-forests-classifier-python)

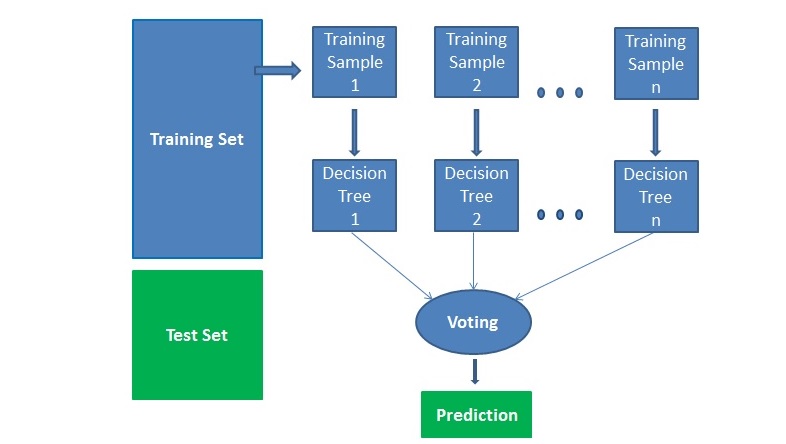


Figure 7 How Random Forest works

### K-Nearest Neighbour

The K-Nearest Neighbouralgorithm has a lazy learner approach. It works with the concept that similar things exist in close proximity. It has no training time because all the computation is moved on prediction phase. At the begin it stores just all the data and during the prediction it finds the K tuples nearest to the instance to predict. In case of classification, it returns the mode of the K nearest tuples. [[9]](https://scikit-learn.org/stable/modules/neighbors.html)

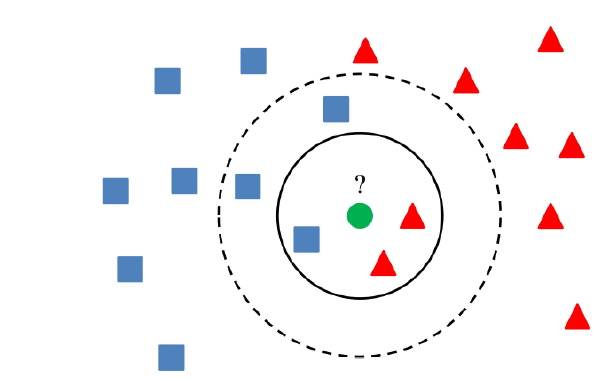


Figure 8 Graphical rappresentation of distance in KNN

### Naïve Bayesian

The Naïve Bayes is a statistical classification technique based on Bayes Theorem. This algorithm is based on the assumption that features are conditionally independent. This simplifies a lot the computation of posterior probability in the Bayes formula:

where:

* is the posterior probability of the hypothesis given the data X;
* is the posterior probability of data X given a specific hypothesis H;
* is the prior probability that hypothesis holds (regardless of data);
* is the prior probability of the data (regardless the hypothesis);

The aim of Bayesian classifier is to determine the class(hypothesis) with highest posterior probability. [[10]](https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn) [[11]](https://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes)

### Support Vector Machine (SVM)

The support vectors machine algorithm is one of the most popular supervised machine learning algorithms. The objective of the support vector machine algorithm is to find the best hyperplane in an N-dimensional space, where N is the number of features, that distinctly classifies the data points.

The best hyperplane is the plane that has the maximum margin, that is the maximum distance between data points of both classes. Maximizing the margin distance provides some reinforcement so that future data points can be classified with more confidence. Support vectors are data points that are closer to the hyperplane and influence the position and orientation of the hyperplane.



Figure 9 Representation of the optimal hyperplane

If the data are not linearly separable it’s necessary to add a new dimension and applying a non-linear separation using a kernel function. [[12]](https://scikit-learn.org/stable/modules/svm.html) [[13]](https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47)

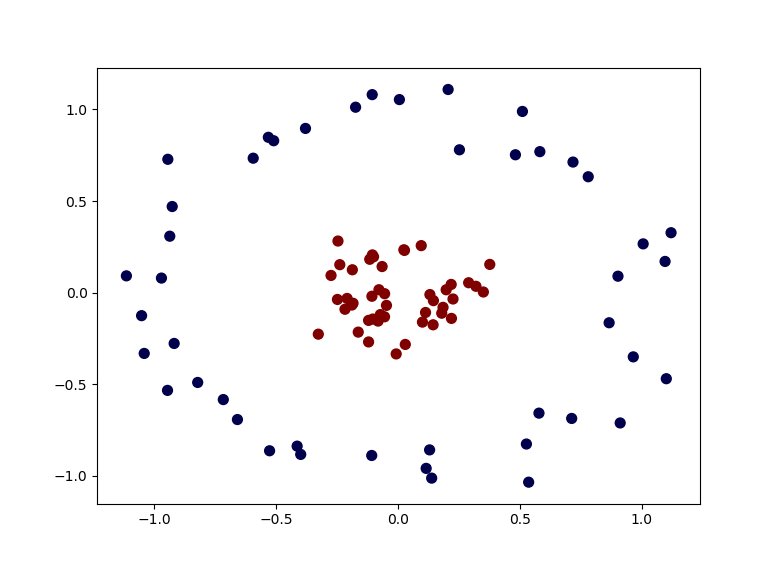


Figure 10 Example of data not linearly separable

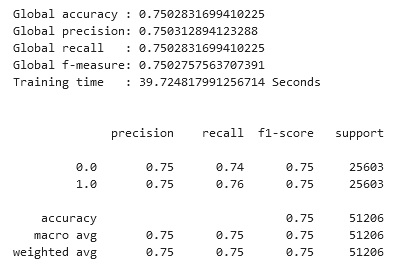
## Evaluations of results

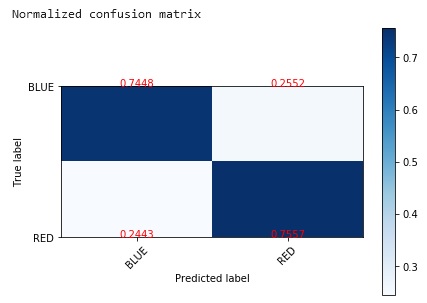
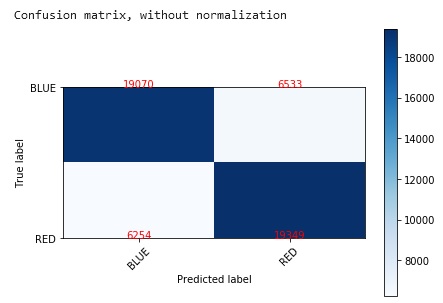
Before applying the data mining algorithms an accurate phase of hyperparameter tuning has been done, in order to evaluate different behaviour of the algorithms and their performances.

**Random Forest**

The random forest algorithm has been executed choosing 10 estimators (number of trees in the forest) and executing in parallel a number of jobs equal to the number of cores of the current machine (6). The metric used for measuring the quality of a split is the *Gini Index*.

The result obtained is the following:



****

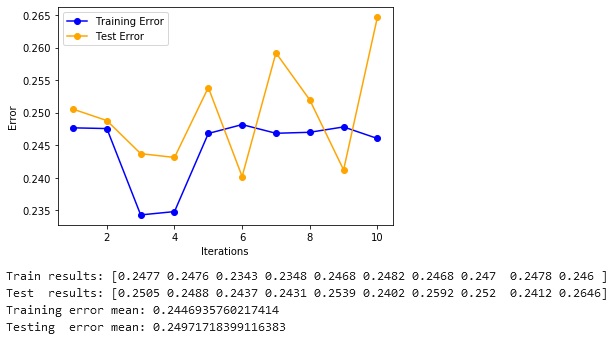
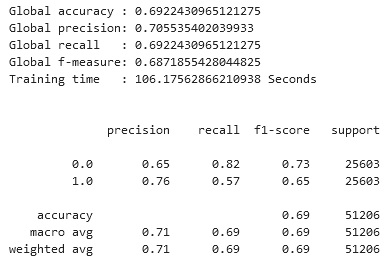
****

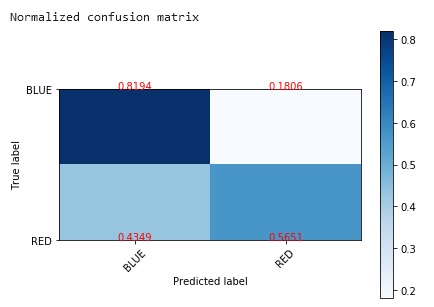
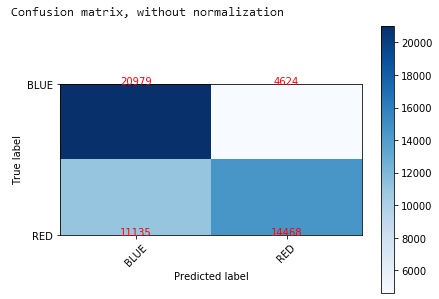
Figure 11 In these pictures: the summarization of the obtained results, the confusion matrix and a graph representation of the mean absolute error in train and test set during the cross validation, using Random Forest as classification Algorithm.

**K-Nearest Neighbour**

The K-Nearest Neighbour has been executed setting the value of k equal to 11. This value has been selected empirically after several tests considering the range of values [1,20] and looking at the accuracy prediction capability per class.

The result obtained is the following:





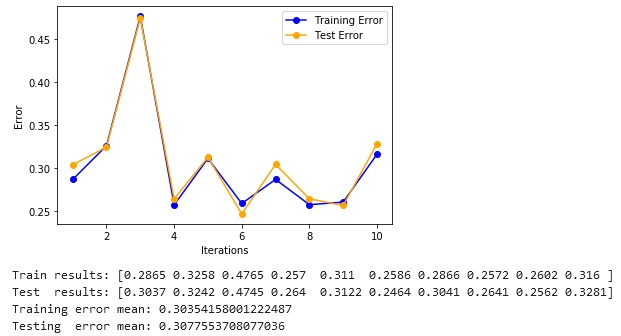
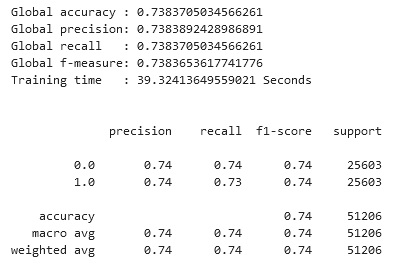


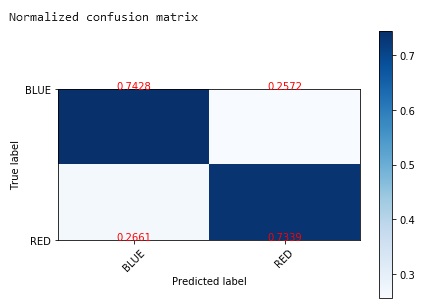
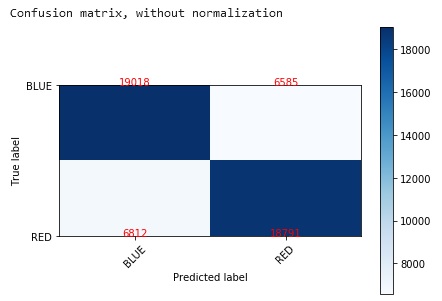
Figure 12 In these pictures: the summarization of the obtained results, the confusion matrix and a graph representation of the mean absolute error in train and test set during the cross validation, using KNN as classification Algorithm.

**Naïve Bayesian**

The Naïve Bayesian has been executed with default parameters and assuming that the likelihood of the features is gaussian.

The result obtained is the following:

****

****

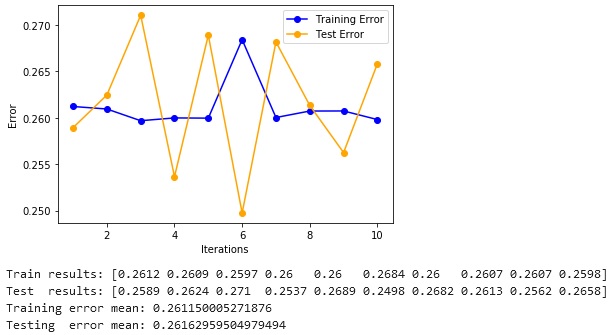
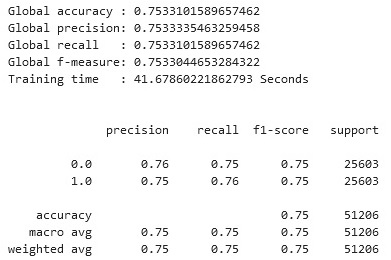
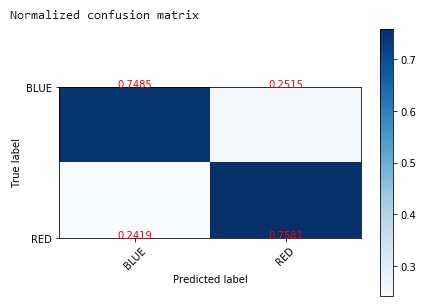
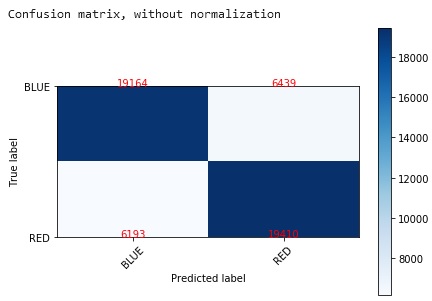
****

Figure 13 In these pictures: the summarization of the obtained results, the confusion matrix and a graph representation of the mean absolute error in train and test set during the cross validation, using Naïve Bayesian as classification Algorithm.

**SVM – Linear Separation**

The SVM algorithm has been executed with default parameters and supposing that the tuples are linearly separable.





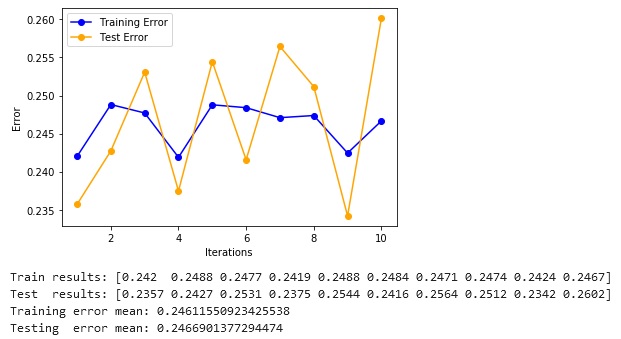
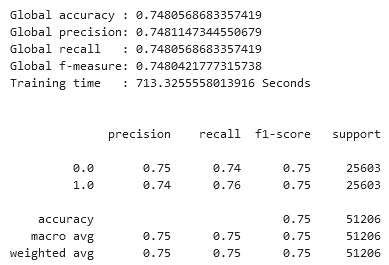
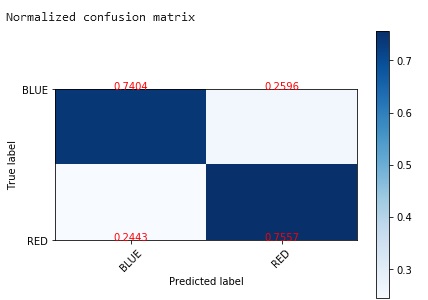
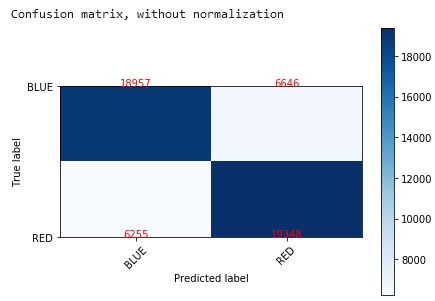


Figure 14 In these pictures: the summarization of the obtained results, the confusion matrix and a graph representation of the mean absolute error in train and test set during the cross validation, using SVM as classification Algorithm.

**SVM - Non-Linear Separation**

The SVM algorithm has been executed assuming that the tuples are not linearly separable, so it has used a polynomial kernel for separation of the two classes.





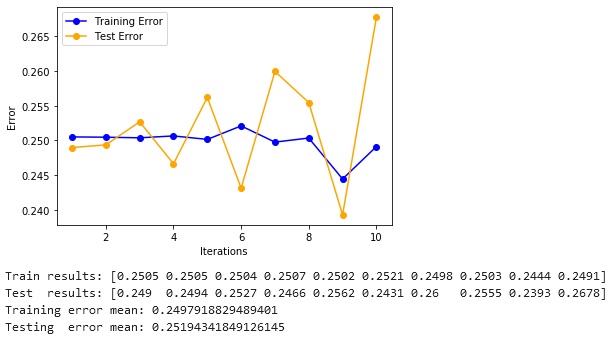


Figure 15 In these pictures: the summarization of the obtained results, the confusion matrix and a graph representation of the mean absolute error in train and test set during the cross validation, using SVM as classification Algorithm.

### Observations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Algorithm** | **Accuracy** | **Precision** | **Recall** | **F-Measure** |
| SVM - Linear | 75,3310 % | 75,3333 % | 75,3310 % | 75,3304 % |
| Random Forest | 75,0283 % | 75,0312 % | 75,0283 % | 75,0283 % |
| SVM - Not Linear | 74,8056 % | 74,8147 % | 74,8056 % | 74,8042 % |
| Naïve Bayesian | 73,8370 % | 73,8389 % | 73,8370 % | 73,8365 % |
| KNN | 69,2243 % | 70,5535 % | 69,2243 % | 68,7185 % |

This table summarize the performances of the various classifier in descending order of accuracy.

The values of accuracy and recall are equal for all the algorithms because the pre-processed dataset contains the same number of instances per class and this is a binary classification problem in which the two classes have the same importance.

In general, the obtained results are similar.

Most of the models do not suffer too much the underfitting phenomena, because filters and hyperparameter have been chosen with the aim of maximizing the accuracy.

The only model that suffers a bit underfitting is the KNN. Looking at its confusion matrix, it is able to recognize the first class with a high accuracy, but not very well the second class.

The Naïve Bayesian recognizes very well both classes and has at the same time a good accuracy score. However, this model is built with the assumption that features are conditionally independent even if some of the attributes are not.

The SVM and Random Forest obtained the highest score of accuracy with small differences. The most evident is the training time. SVM Not Linear is slower respect Random Forest and SVM – Linear. In particular the training of SVM Not Linear model is computationally heavier respect to Linear SVM due to the construction of kernel function.

The phenomenon of overfitting is limited in all the algorithms used, thanks to the use of cross validation. The values of the mean square error on the train set and on test set are approximately the same.

A paired t-test has been done in order to understand if there is a statistically difference between the two algorithms with better accuracy, assuming that the mean square error distributions are normally distributed and the variances of the two population are not reliably different.

****

****

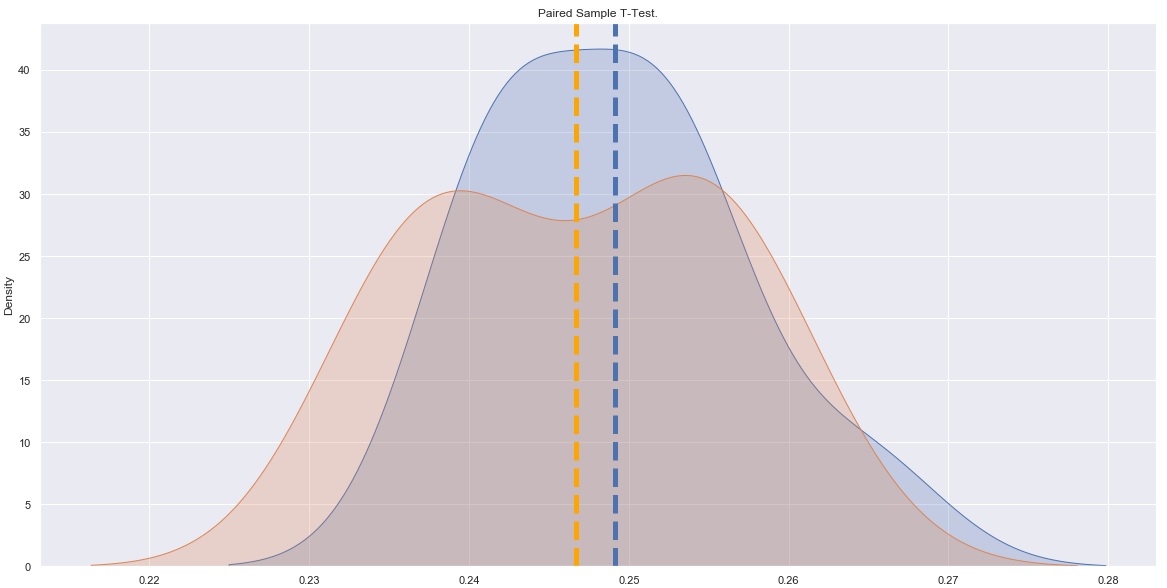
****

Figure 16 Distribution of the two test-errors in Random Forest and SVM Linear

****

The result of the t-test shows that null-hypothesis can’t be rejected and so the two distributions are not statistically different. [[14]](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html)

### Conclusions

SVM algorithm with linear separation is chosen as model for future match prediction. Making a comparison between the two algorithms (SVM – Random Forest), the SVM has lower difference between training error and test error, this means better generalization capabilities. In general, for a two-class classification problem a SVM approach is more suitable to recognize data that tends to be linearly separated like in this case.

The obtained result is quite good. It would be improved considering more instances in the training dataset, evaluating other types of classification algorithms and considered different attribute selection methods.

# **Application**

## Analysis Phase

**Description**

This application is designed for people who are interested in League of Legends game. A user can insert information related to a League of Legends match at 10 and 15 minutes and get a possible outcome of that game.

**Main actors**

There is only one actor: *generic user*. He/she is the person who want to predict the team winner. There is no registration phase and no other information related to the generic user are needed.

**Scenarios**

There is only one scenario:

*Match's Information Insertion Scenario*, where the system gives the possibility to insert the specific match characteristics at minute 10 and 15 for both participating teams. These information are the same analysed during the building of the raw dataset.

**Requirements**

*Functional Requirements*: the application must treat information related to League of Legends match in order to predict a possible team winner. The application must implement a functionality to load match’s information from external source in order to simplify the user insertion information operation. There must be a reset function in order to clean the previous insertions and prediction. The outcome prediction must be clear and well visible.

*Non-Functional Requirements*: the application must be portable, intuitive and easy to use. The application must embed a data mining classification algorithm in order to make a prediction. The user shall not wait too much time during the prediction phase. The result of the prediction should be most accurate as possible and the trained model should adapt well to new unseen data. The code must be readable and modulated because in this way it’s easier to add new functionalities and to resolve eventual errors.

**Use Case Diagram**

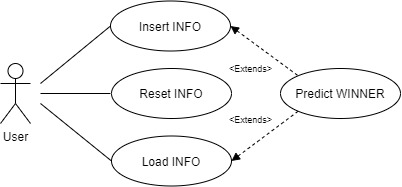
****

Figure 17 Basic use case diagram

## Design Phase

In this phase there is a short description of the application’s main scenario. Pseudo-code and mockup are used to explain and visualize better the behaviour.

**Match's Information Insertion Scenario**

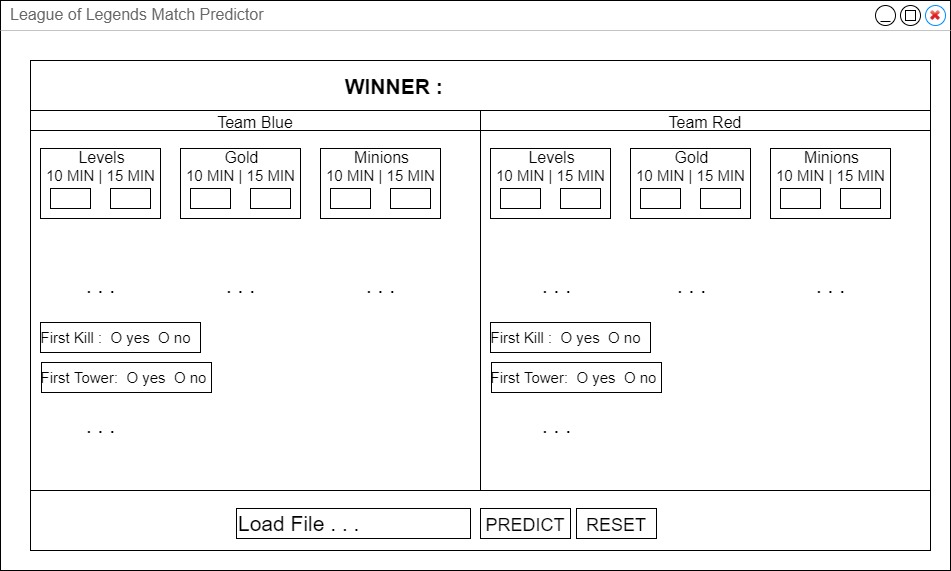
****

Figure 18 Mockup of the web application

1. The *system* shows the information fields related to the two teams and to the time interval at minute 10 and 15
2. The *user* can insert manually all the information
3. IF *user* clicks on Load File
   1. The *system* shows the file system path
   2. IF *user* selects a file
      1. The *system* reads the loaded file and populate the information fields
4. IF *user* clicks on Reset
   1. The system clears all the information fields
5. IF *user* clicks on Predict
   1. The system pre-processes the new data and try to predict the outcome of the match and shows the result on the top of the page

**UML Diagram**

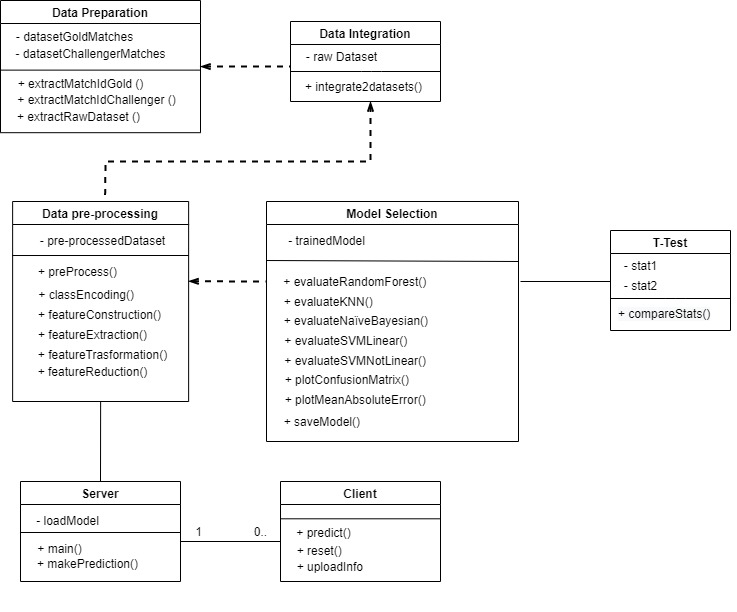
****

Figure 19 Diagrammatic representations of software components

**Application Architecture**

In order to satisfy the non-functional requirement of portability and simplicity, the choice of a (client-server) web application was considered a good solution. The server embeds the trained classification model. The client sends HTTP requests with information about the match. The server receives the request, pre-process the data received and then predict the outcome. At the end, it responds to the client with the prediction.

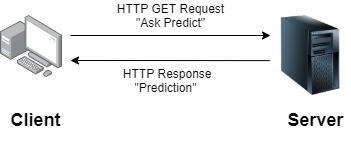


Figure 20 Client-Server application

## Implementation phase

The choice of using Python as programming language is nice because it offers lot of dynamic libraries and modules that help in handling raw data in the pre-processing phase and allows modularization of the code. Thanks to the Sklearn library it’s possible to use several classification algorithms and implements techniques like Cross Validation in few lines of code. It also allows the use of Mathlab libs for plotting results and visualizations of the outcomes.

The web application is built using Flask, a micro-framework written in Python. It gives developers flexibility and permits to build a web application quickly using only a single Python file.

The developing of GUI is simple too. The web interface is an HTML page where all the components are standard and easy to use.

The project is organised in several folds:

* **dataset**, it contains the dataset files in csv form. These are obtained using the scripts in \scripts\datasetBuilding following the same method present at *Dataset Building* paragraph.
* **scripts**, it contains all the python scripts used for this project. For each python file ends in ‘.py’ there is the corresponding ‘.ipynb’, for the execution of the code using Jupyter Notebook.
  + PreProcessingRawData.py, this script contains all the necessary operations to make the file in a proper form for data mining phase. This file is used in training but also for the prediction of new data.
  + ModelSelection.py, this script is located in a sub folder named datamining and contains the building of the filter with the pipeline, the implementation of Cross Validation method and the computation of various metrics for the evaluation of results. At the end it also saves, the model just trained on a different file in order to permit to load it in a different module avoiding losing training time at each server start up.

The rebalance filter has been applied as first filter, outside the main pipeline, because it belongs to a different source library *imblearn.py* and it gave error during the execution.

Here the most code relevant part:

#Filters

treeDiscr **=** DecisionTreeDiscretiser **(**cv**=**10**,** scoring**=**'accuracy'**,** variables**=[**'diffTotGold'**,**'diffMinions'**,**'diffDomRatio'**,**'diffJglMinions'**,**'diffWardVision'**,**'diffTotGoldAvailable'**],**regression**=False)**

# param\_grid={'max\_depth':[15,20,25,30],'min\_samples\_leaf':[500,1000,5000]})

scaler **=** MinMaxScaler**(**feature\_range**= (**0**,**1**))**

feat\_selector **=** SelectKBest**(**chi2**,** k**=**10**)**

#Data-mining algorithms

#dmAlg=RandomForestClassifier(n\_estimators=10,n\_jobs=-1,criterion="gini")

#dmAlg = neighbors.KNeighborsClassifier(n\_neighbors=11)

#dmAlg = GaussianNB()

dmAlg**=** LinearSVC**()**

#dmAlg=svm.SVC(kernel='poly')

pipe **=** sklearn**.**pipeline**.**Pipeline**([**

**(**"treeDiscr"**,** treeDiscr**),**

**(**"scaler"**,**scaler**),**

**(**"feat\_selector"**,**feat\_selector**),**

**(**"dmAlg"**,** dmAlg**)**

**])**

predicted\_targets **=** np**.**array**([])**

actual\_targets **=** np**.**array**([])**

kf **=** KFold**(**n\_splits**=**10**)**

i**=**0

start\_time **=** time**.**time**()**

list\_training\_error **=** **[]**

list\_testing\_error **=** **[]**

**for** train\_index**,** test\_index **in** kf**.**split**(**X**):**

i**=**i**+**1

X\_train**,** X\_test **=** X**.**iloc**[**train\_index**],** X**.**iloc**[**test\_index**]**

Y\_train**,** Y\_test **=** Y**.**iloc**[**train\_index**],** Y**.**iloc**[**test\_index**]**

rus **=** RandomUnderSampler**(**replacement**=False)**

X\_res**,** Y\_res **=** rus**.**fit\_resample**(**X\_train**,** Y\_train**)**

X\_train**=**X\_res

Y\_train**=**Y\_res

pipe**=**pipe**.**fit**(**X\_train**,** Y\_train**)**

In the same sub folder, there is also the script *T-test.py* used for analysing if there is a statistical difference between the two classifiers.

# Predict the labels of the test set samples

predicted\_labels **=** pipe**.**predict**(**X\_test**)**

# Predict also the train samples in order to calculate the mean\_absolute\_error on train set

y\_train\_data\_pred **=** pipe**.**predict**(**X\_train**)**

fold\_training\_error **=** mean\_absolute\_error**(**Y\_train**,** y\_train\_data\_pred**)**

fold\_testing\_error **=** mean\_absolute\_error**(**Y\_test**,** predicted\_labels**)**

list\_training\_error**.**append**(**fold\_training\_error**)**

list\_testing\_error**.**append**(**fold\_testing\_error**)**

#Append the results obtained at one iteration

predicted\_targets **=** np**.**append**(**predicted\_targets**,** predicted\_labels**)**

actual\_targets **=** np**.**append**(**actual\_targets**,** Y\_test**)**

**print(**"Cross validation "**+**str**(**i**))**

**print(**accuracy\_score**(**Y\_test**,** predicted\_labels**))**

**print(**classification\_report**(**Y\_test**,**predicted\_labels**))**

stop\_time**=**time**.**time**()**

* **static**, in this folder there are all the files used by the web application on client side. It contains the JQuery scripts for handling user interaction and css file to apply styles on the web interface. In particular in *jqueryUserInteractionFunctions.js* there are methods for the population of the input fields during the loading of external match file and methods for the building of an AJAX request.
* **template**, it contains just the *index.html*, the only user interface of the web application.
* **test**, it contains 10 files of new matches used in the test phase, one instance per file.

Finally, there is the *Server.py* script fundamental for launching the web application. At the begin it first load the saved model in *script* folder then it tries to start the server. The server is able to handle 2 types of requests. First, it returns the web interface *index.html*  when the user sends the first request, here is the snipped of the code:

@app.route**(**"/"**)**

**def** main**():**

**return** render\_template**(**'index.html'**)**

Second, it handles the prediction request. When a prediction request arrives, the server first create and pre-process a new instance with the values arrived in the request, then it uses the model loaded at the begin to predict the outcome and returns it as a JSON response. In order to have an easy debug of the application, 2 csv files are created in this folder with the values arrived from the requests: *rowInstance.csv*, *preProcessedInstance.csv*. Here is the snipped of the code:

# "http://127.0.0.1:5000/makePrediction?&params

@app.route**(**'/makePrediction'**,**methods**=[**'GET'**])**

**def** makePrediction**():**

newRowInstance**=**readParams**(**request**)**

newInstance**=**preProcess**(**newRowInstance**)**

newInstance**.**to\_csv**(**'preProcessedInstance.csv'**,**index**=False)**

newInstance**=**pd**.**read\_csv**(**'preProcessedInstance.csv'**)**

X\_DataTest**=**newInstance**.**loc**[:,** 'diffLevel'**:]**

yResult**=**pipe**.**predict**(**X\_DataTest**)**

**if** **(**yResult**[**0**]==**0**):**

answer**={**"winner"**:**"BLUE"**}**

**else:**

answer**={**"winner"**:**"RED"**}**

**return** jsonify**(**answer**)**

## Test phase

New League of Legends matches are played and ten of them have been selected randomly and used to test the application. They are located in the *test* directory. The prediction phase is quick and it requires less than a second.

8 of 10 matches are correctly classified. The tests incorrectly classified are:

* test number 5, the correct prediction would be Team Blue
* test number 8, the correct prediction would be Team Red

80% of test accuracy can be considered a good result. It reflects in part the value of accuracy of the model (75,3310%) discussed above. It would be better considering more than 10 test cases in order to have a truthful result.

# **User manual**

## Preliminary Actions

The first thing to do is to run the web server.

Open the *Command Prompt* on the current project directory, then type the following command:

.\Server.py

If errors not occur a new console window will open.

Take in mind the address where the Python server is running. This address is used to have access to the application.

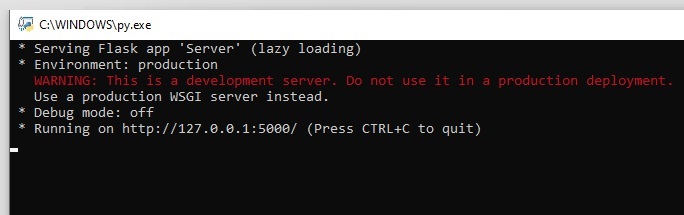


Figure 21 Screenshot of starting server's output

## How to use the application

To use the application, it’s necessary to have a web browser and typing on the search bar the address indicated at the previous point, then start the research.

The web browser shows the web application interface.

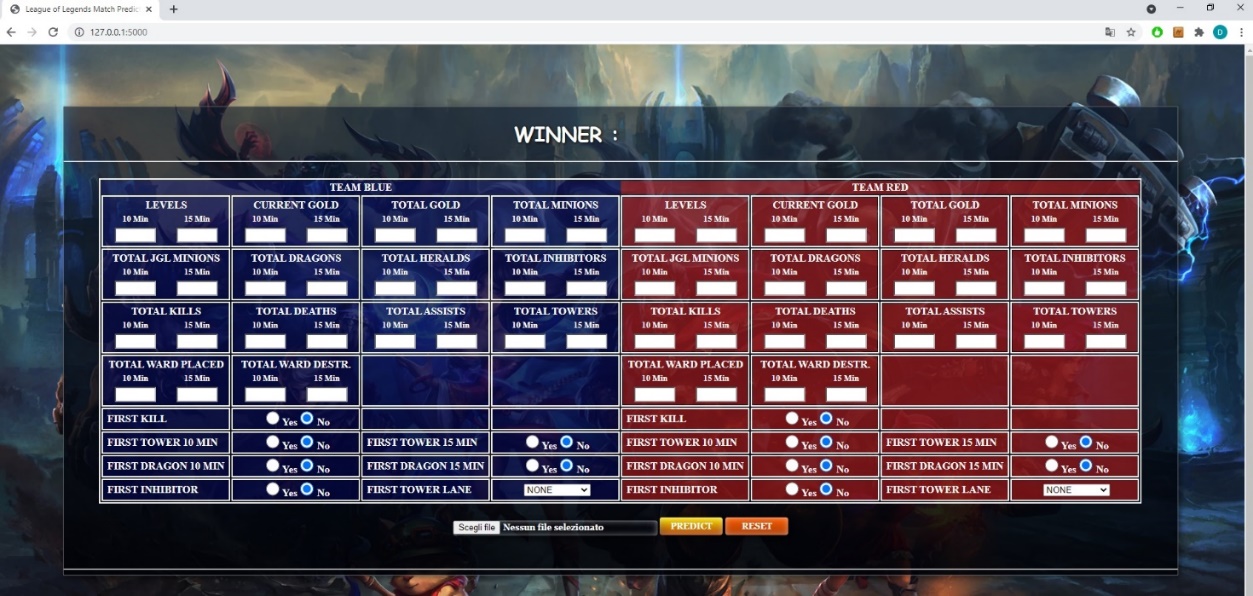


Figure 22 Screenshot of the main web interface

The user can fill manually all the information related both team at 10 and 15 minutes otherwise if he/she arranges a csv file in appropriate form, he/she can load it pressing the button “Load File”. Once loaded, the application will fill automatically all the input fields.

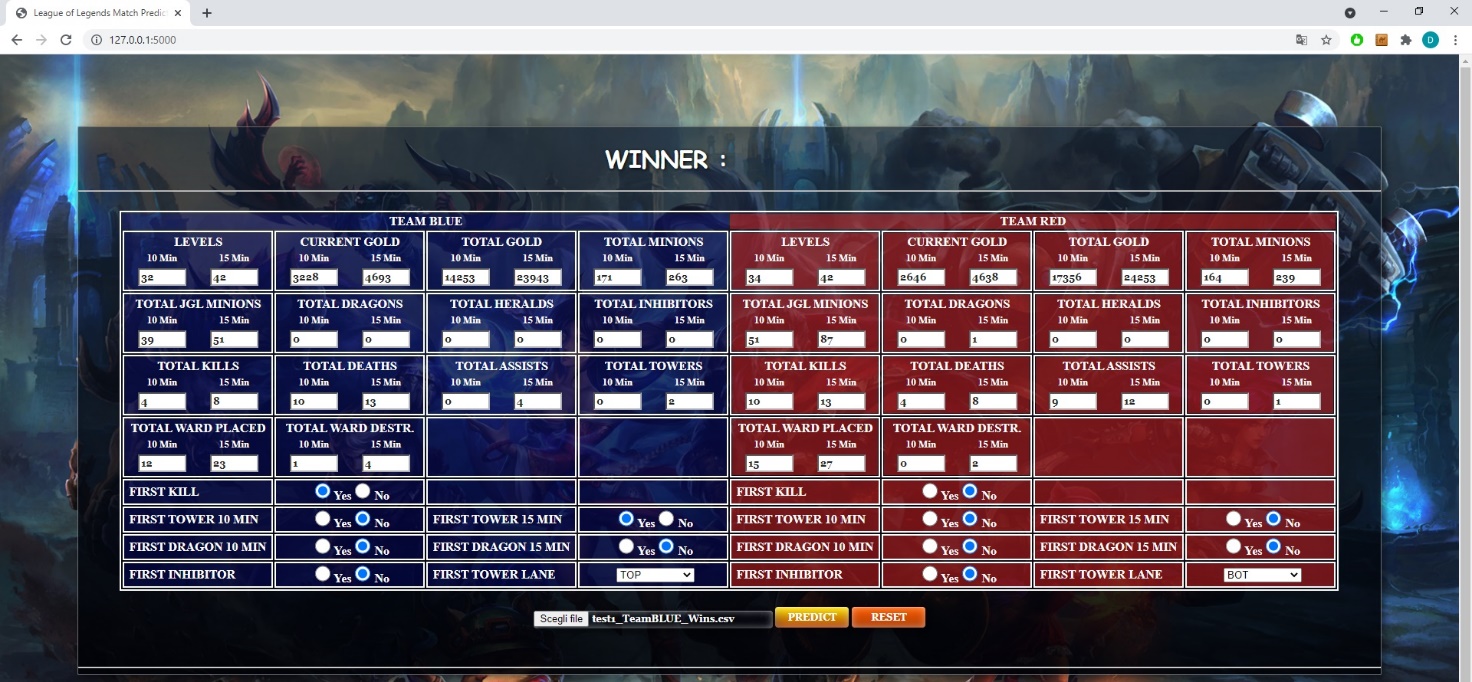


Figure 23Screenshot of the populated web interface

The user can click on *Predict* button, then the application will predict which of the two teams will win. If the users want to clean all the fields, he/she can click on *Reset* button.

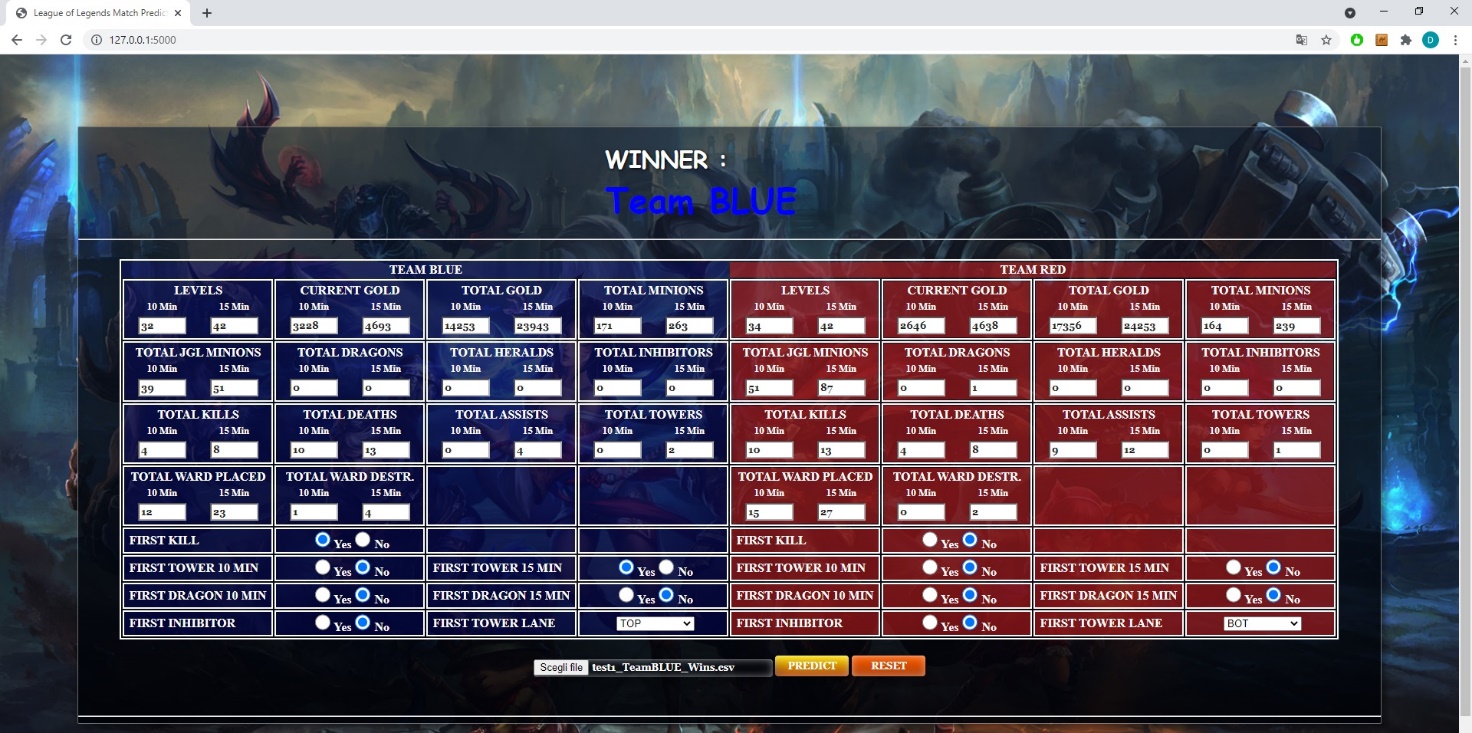


Figure 24 Screenshot of the predicted team winner

**References**

[1]: <https://developer.riotgames.com/>

[2]: <https://www.kaggle.com/fernandorubiogarcia/league-of-legends-high-elo-patch-1016>

[3]: https://imbalanced learn.org/stable/references/generated/imblearn.under\_sampling.RandomUnderSampler.html

[4]: <https://feature-engine.readthedocs.io/en/1.0.x/discretisation/DecisionTreeDiscretiser.html>

[5]: <https://towardsdatascience.com/an-introduction-to-discretization-in-data-science-55ef8c9775a2>

[6]: <https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html>

[7]: <https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html>

[8]: <https://www.datacamp.com/community/tutorials/random-forests-classifier-python>

[9]: <https://scikit-learn.org/stable/modules/neighbors.html>

[10]: <https://www.datacamp.com/community/tutorials/naive-bayes-scikit-learn>

[11]: <https://scikit-learn.org/stable/modules/naive_bayes.html#naive-bayes>

[12]: <https://scikit-learn.org/stable/modules/svm.html>

[13]: <https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47>

[14]: <https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.ttest_ind.html>

[15]: <https://towardsdatascience.com/is-your-model-overfitting-or-maybe-underfitting-an-example-using-a-neural-network-in-python-4faf155398d2>

All project material is available on:

<https://github.com/AlterVigna/League-of-Legends-match-predictor>