Football Results Predictions for the Major 6 European National Leagues

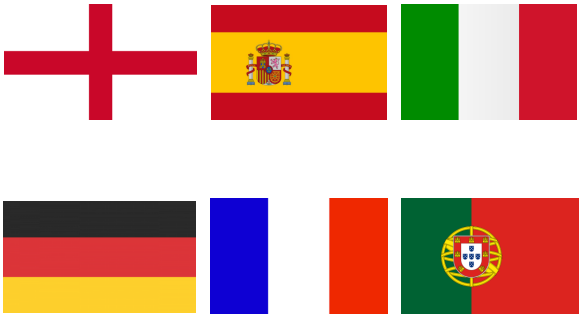

Table of Contents

Introduction ***3***

Methodology ***4***

Dictionary ***5***

Concepts 5

Variables 5

Technology used ***7***

Structure ***8***

Notebooks ***9***

1. **Table Results** **9**
2. **Table Fifa** **11**
3. **Merge Results and Fifa tables** **12**
4. **Find Best Model** **13**
5. **Apply Best Model** **15**

5.1 Apply Best Model (PY file) ***17***

Streamlit_app_GCS.py ***18***

Using Cron to automate the predictions update ***19***

Streamlit Web App for the end user ***20***

Conclusion ***21***

Introduction

The objective of this project is to build a prediction model that is able to forecast with the highest accuracy possible the result of the national leagues football matches from the top 6 European leagues: English, Spanish, Italian, German, French and Portuguese.

The prediction will be based on the calculation of the probability of each one of the possible outcomes of a match, in a classic *1 X 2* classification: *1* means a victory for the home team, an *X* means a draw and a *2* means a victory for the away team. The class with the highest probability will be the final prediction for the respective game.

In general football results have an obvious level of uncertainty: weekdays (matchdays) without surprises are rare, as not always the best team wins. In fact, there is a variety of factors that could impact the score of a game. Considering complexity, data availability, time constraints and factor relevance, the ones I chose for this project are:

- Team ratings → based on the FIFA Video game criteria
- Match location → who plays at home
- Team fatigue → how many games have the team played in the previous 21 days
- Team season form → how many points the team got in the season so far
- Team recent form → how many points the team got in the previous 5 games
- Game's history → what was the outcome of the same game in the past years
- Budget each one of the teams has available
- Rivalry between teams

The project only predicts the matches for the current and following weekdays. That is because for longer time windows the predictions would be even less reliable, as some factors depend on the recency of results. For the prediction, we train the algorithm with all the results from 2005 until today.

In all the notebooks you will constantly find comments explaining what is being done. Hopefully it will help the reader to understand the steps taken and easily follow the logic of the code.

Also, please note that this project demanded a great effort in terms of data processing and cleaning. I didn't include in the scripts every detail of the work I developed to found everything that needed to be corrected after the data extraction; instead, I have preferred to directly include the corrections to what was wrong/misplaced. Anyway, you will find comments in the notebooks that describe these situations.

Methodology

There are a lot of webs with data that cover football: newspapers, clubs, official football associations, betting websites, etc. After analyzing many available websites, I decided to scrap the web of https://www.resultados-futbol.com/ to extract all the results from the current and past seasons. Apart from having enough information there to calculate the fatigue, season form and recent form, I can also get the match location, the game's history and of course what matches will be played in the current and following weekday. Then, I also scrapped the web https://www.fifaindex.com/ which allowed me to get the team ratings, team budgets, and also the rival team of each one of the teams. Fifa index website only has data from 2005, so all the info extracted from both webs are from that year until the present.

After extracting and cleaning all the data from the 2 sources, I merged the tables into one, so all the info is gathered in one single dataframe. I used this resulting dataframe to train different estimators, evaluate its performance, and find the best one. Later, I used it to calculate the probability of the different outcomes of the current and following weekdays. At last, I created a simple interface using Streamlit where anyone can access to consult those predictions.

In many of the cells I had used the magic command *%%time* in order to calculate the time needed until the cell has completely finished to run. I do that because many of the cells take significant amount of time (over an hour in the case of one of the web scrapping cells), and as I have run them many times during the project, this helped me to manage my work. That is why I had added cells to download the resulting dataframes into excel files: thanks to it, in the following day I could just read the excel instead of running the cell again. Besides, I could also read those files in other notebooks.

I tried to make this project as replicable as possible. In this sense, with just a few adjustments, it should be valid to forecast not only the weekdays of the current season, but also the ones of a future season in which an end user could be requesting predictions.

At the top of each notebooks, you will find what libraries you need to import for that respective notebook to run. You will also see that I use code to turn off the warning relative to setting with copy. I do that because in many of the cells I get that warning, and after reviewing every case in which it happens, it does not affect the outcome of the code that I need. So, to make the notebooks look cleaner, I deactivate the warning.

You will see that in some cells I create dataframes called *df_aux*. I normally give that name to *auxiliar* dataframes I create in order to get to a final result, hence the name *df_aux*.

Dictionary

Concepts

Weekday → means matchday. A weekday is the number of the round of its respective league. All the 6 leagues have its teams playing each other twice in a season (one game at home and another away). The number of weekdays of a league is equal to the number of teams of that league minus 1 multiplied by 2 (a league with 20 teams would have 38 rounds, or 38 weekdays).

Table_results_... → all dataframes whose name starts with *table_results* means that the data in it was extracted from the the website https://www.resultados-futbol.com/.

Table_fifa_... → all dataframes whose name starts with *table_fifa* means that the data in it was extracted from the website https://www.fifaindex.com/.

Variables

Table_results_previous_years_original → dataframe that shows the info extracted from the site resultados-futbol.com for all the previous years (from 2005 to the current year, excluding the current year), with little to none editing.

Table_results_previous_years_edited → dataframe I get after performing many edits to the *Table_results_previous_years_original*.

Table_results_current_year_original → dataframe that shows the info extracted from the site resultados-futbol.com for all the current year, with little to none editing.

Table_results_current_year_edited → dataframe I get after performing many edits to the *Table_results_current_year_original*.

Table_results_all_years_edited → dataframe that combines both *Table_results_previous_years_edited* and *Table_results_current_year_edited*.

Table_fifa_all_years_original → dataframe that shows the info as extracted from the site fifa-index.com for all the years (from 2005 to the current year), with little editing.

Table_fifa_all_years_edited → dataframe I get after performing many edits to the *Table_fifa_all_years_original*.

Merged_table_from_2005 → dataframe I get after merge the *Table_results_all_years_edited* and *Table_fifa_all_years_edited*. This will be the base to train different machine learning models.

Table_final_user → dataframe that includes the games and the predictions and that will be shown to the end user in the Streamlit web app.

Technology used

The programming language I used in this project is Python, and I have used Jupyter notebooks to run the scripts. For that, I used Anaconda to manage my virtual environments. I created a new environment with the python version 3.9 using the terminal on my Mac. Then, on the Anaconda app, the first thing I installed on that new environment was the package Jupyter, so I could work on the notebooks with it.

You will find on the first cell of each notebook all the libraries and dependencies you need to run the whole notebook. These are the libraries you will need to have installed on your environment to run all the code (you can install them using the command pip install): *requests, bs4, pandas, fuzzywuzzy, python-Levenshtein* (in this case, to remove the warning "Using slow pure-python SequenceMatcher"), *html5lib, lxml, openpyxl, matplotlib, sklearn, seaborn, google-cloud,* and *google-cloud-storage*. However, I have also included in the repo an yml file (*environment_tfm.yml*) that you can use in order to replicate the environment I used to run all the notebooks. If you have Anaconda installed on your machine, in this page you can find instructions to see how you can create the environment with the file.

I also created a bucket in Google Cloud Storage where I host the final table with the predictions of the matches results. If you need more info about how to create a bucket, you can check this article from Google support.

Another software I used was cron. You will need to install it (or use an equivalent software) that allows you to schedule jobs in your machine. I use it to automatically update every day at 11am the data in the table the algorithm uses to learn from and calculate the best prediction per match, as every weekday generates new information that should be added to the table. I decided to do the update every day because the file only takes about 12 min to run, and there are many league games that are played during the week, and not only on weekends. By the way, I could also use a remote machine to run the update instead of my own machine, but considering I use my Mac every day to work and the file runs quickly without interfering with the performance of my laptop, I decided to do it directly on my laptop.

Finally, I used Streamlit to build the interface that everyone can access to consult the predictions done by the developed model in this project. This app basically reads the predictions dataframe that is stored in Google Cloud Storage. This way, I assure the end user can have a fast access to an updated predictions table.

If you want to know more about Streamlit, in this webpage you can find the instructions to guide you through the installation. There you will also find useful links to learn how to create your own app.

Structure

I divided the code in 5 different jupyter notebooks:

1. Table Results
   In this notebook, I start by extracting all the needed info from the website https://www.resultados-futbol.com/. It includes the results and the dates of all the matches for the teams playing the 6 leagues in question (games of the national leagues and other competitions). With the dates, I calculate the fatigue and with the results I calculate the season form and recent form, and the game's history. I also clean and prepare all the data to be used later for the training of the algorithm.

2. Table Fifa
   Here I get data from https://www.fifaindex.com/. From this site, I get the teams ratings, the budgets and the respective rival teams of each team. I also clean it and prepare the info, just like in the previous notebook.

3. Merge Results and Fifa tables
   In the 3rd step I join both tables, so I get what we need to train the algorithm in only one table.

4. Find best model
   With all the info in one dataframe, I apply different models/estimators with different parameters to find the one that works best.

5. Apply the best model
   To finish, I apply the best model to the current and following weekday of each one of the national leagues and we finally get the predictions we wanted.

There is a 6th file, with a script also written in Python (PY file) that I run automatically everyday which is the one that generates the dataframe with the predictions that is stored in my Google Cloud account.

Let's go in more detail to each one of these notebooks and files.

Notebooks

1. Table Results

This is the first notebook, and I start the project by webscrapping the page https://www.resultados-futbol.com/. This is where I get all the results from the 6 leagues for every year from 2005 until today.

There are 2 important points to mention:

- I decided to webscrap separately the current year results from the previous seasons. That is because there are a lot of years from 2005, and as I need to constantly update the current year table with the results from every weekday, it would take a lot more time and resources if I had to extract all the years info every week.
- The first year that is extracted (2005) is hard coded because the other website (Fifa index) only has data from that year. However, the current year is not hard coded so the code can easily be applied with minimal adjustments in future seasons.

This notebook is divided in these sub sections:

• Create table_results_previous_years

Here you can find all the code to extract the results I need. This is possibly the cell that takes longer to run, so I had added some text to be printed during the process, so I can follow the status. I also had added a cell in which I extract the output to an excel file in my machine, so I could read from it instead of running the cell every time I opened the notebook.

• Create functions to manage the table

In the previous step, I basically only extracted the raw data. Then, in this phase I define the functions that allow me to edit the info in the table, and also do some calculations that will set the foundation to compute part of the factors I will have in account to train the model for predictions, such as the team fatigue, form, game history, etc. I also need to mention that I decided to create functions and not apply directly the code to the raw table because there will be another web scrapping task later to extract current year's results, and this way I can write the functions code only once and apply it twice.

• Apply the functions to the table_results_previous_years

Here I just apply the functions to the table with the results from past years. At this moment, I only apply the *edit_raw_table, fatigue*, and *points_respective_year_and_last_games* funtions. The last one (*points_between_teams*) I only apply later, after I join this table with the table_results_current_year, as I would have to do it later anyway (if I would apply this function individually per table, it would not be correct for the table_current_year_results, as I need past data for the calculation). And as these cells take some time to run, again I create a new excel file at the end of this subsection.

- Create table_results_current_year

Finally, I generate here the data with the results of the current year. I generate it the same way as for the table_results_current_year, with the exception that this time is just for one year, and so the cell runs much faster.

- Apply the functions to the table_results_current_year

Again, the last function *points_between_teams* is not going to be applied now but later.

- Correct team names of table_results_current_year

I noticed that a few teams have a different name for previous years and current year in the https://www.resultados-futbol.com/, and I need it to be the same so later the algorithm can work with all the data based on team names. For this I use the library wuzzyfuzzy. In the cell you have the explanation to how it works.

- Join Tables previous years and current year

Here is where I put together the 2 tables.

- Apply function points_between_teams

This is the last step, in which I can finally apply the last function, as both tables had already been joined. This is where we get the table_results_all_years.

2. Table Fifa

Here we extract all the ratings, budgets and rival teams per each one of the teams in the different national leagues, from 2005 until the current season, from the site https://www.fifaindex.com/.

These are the sub sections of this notebook:

- Create table_fifa_all_years

  In this step, I extract the ratings, budgets and rival teams.

- Add missing teams information

  Although the fifa index website has a specific page per team, per year and per country, I noticed that for some years and leagues, not all of them are present in the Fifa index league table. Considering this is what I use to extract all the fifa teams in the previous sub section, I need to take action, otherwise, there would be teams with missing information. To solve this, I created some code in order to find those teams missing, look for their information, and add it to the table_fifa_all_years. It is very important that all teams are present in this dataframe, so later we can join it with the table_results_all_years.

- Add columns Team-ID and Fifa_team_all_names

  I created a couple of columns that will be important in the following notebook, when we merge the Table_results with the Table_fifa.

3. Merge Results and Fifa tables

The third notebook is the one in which we join both results and fifa tables. Plenty of the names of the teams in the table_results and in the table_fifa do not match. In fact, many of them have a different name in both tables. Example: in one page a club can be called *R. Sociedad*, while in the other it could be *Real Sociedad*. Actually, one same page can have different names for the same team, in different years. This complicates the action of merging the tables.

These are the 3 sub sections of this notebook:

- Create dictionary to match team names

After reading the table_results and table_fifa, I create a dictionary in which the keys will be the names of the teams in the table_results and the values will be the names of the teams in the table_fifa. As there are a lot of teams, I used the library fuzzywuzzy to get some help with the matching. Still, I had to do a few manual corrections, as it is explained in the notebook.

- Join Results table and Fifa table

Once the dictionary is created, I have what I need to merge both tables.

- New column: Rivals

The last step here is to add a column which indicates if the respective match is between 2 rival teams or not.

4. Find Best Model

Once we have our dataframe with all the data, we can create different models with different estimators to see which one gives us the best predictions. We are predicting if the outcome of a match will be *1* (home team victory), *X* (draw between the teams) or a *2* (away team victory), which makes this project to fall under the category of a machine learning multiclass classification problem.

These are the subsections you will find:

- Prepare the dataset to apply different models

In this step, we load the data we generated in the previous notebook, drop a few columns that do not add value to the results predictions and check the correlation between the different features: the objective is to find and drop columns that have a high correlation and dependency with each other, so we make sure we don't add unnecessary complexity nor bias to the models that will be tested. For example, I decided to exclude the ratings of attack, midfield and defense (Home/Away_team_ATT, Home/Away_team_MID, Home/Away_team_DEF) and use instead only the overall rating (Home/Away_team_OVR). I also decided to exclude the feature that shows the budget of the team, as it is quite correlated with the overall rating of the team again.

Then, I split the data in X_train, X_test, y_train, y_test and later I analyze the type of data in each column. There are only 3 categorical variables (Country, Home_team and Away_team), and 12 numerical. However, I decided to consider 2 of the numerical variables (Year and Rivals) as categorical, because although they show numbers, they work similarly as text: showing a higher or lower value does not mean it is better or worse.

- Training and evaluation of the models

In order to avoid some warnings that are displayed when we fit the models, I start by adding a line of code to prevent it. Then, as I want to test different estimators, I decided to create a pipeline that consists of applying 2 transformers as a data preprocess step (Standard Scaler for the numeric features and OneHotEncoder for the categorical ones) and also a model. Then, I chose a few models that are suitable for a multiclass classification problem:

- Dummy Classifier, official documentation here.
- K-Neighbors Classifier, official documentation here.
- Linear Support Vector Classification, official documentation here.
- Decision Tree Classifier, official documentation here.
- Logistic Regression, official documentation here.
- Random Forest Classifier, official documentation here.

The Dummy Classifier (with the strategy "most_frequent", which in this case means it always predicts that the home team will win the match) is used only to have a reference and compare the performance of the other models to it.

You will notice I also added a cross validation step with 5 folds. This will give us a more reliable result of how well each model can predict the final result.

In terms of metrics, in order to evaluate how well a model can predict results, although I start by comparing accuracy and F1 score, I decided to ultimately use the F1 (with average "weighted"). I did that because it seems to me a solid indicator of a model performance, as it takes in account both precision and recall. I added the text "no tuning" to the dataframe name that shows the results of each model because we are running them with the parameters they have defined by default. In the next subsection we will tune them in an attempt to improve our forecast.

- Optimizing models parameters

In the previous steps we could see and compare the results of different estimators. There is no model that showed to work much better than the others, and the F1 score was somewhat similar. So, I decided to give it a try to each one of them and try different parameters to see if we could improve their performance and reach a better result.

In each model you will see what parameters I have tested. In each one of the cases, I selected the ones that have the most impact, according to different articles written by data scientists that I found on the web.

- Best model

After testing different parameters per estimator, we can see that some models have improved more than others. For example, the decision tree model has improved around 0.03 and is now among the best. However, and although the difference is not high, the best one keeps being linear regression (with the *C* parameter set to 10), so it will be the one that we will use to predict results.

5. Apply Best Model

In this last notebook, I apply the best model found previously and apply it to the data, so we can have the best predictions. I have done this notebook in a way that it is the only one I need to run when I want to update the matches and predictions. So, it is kind of a summary of all that was done until this step. These are the sub sections:

- Create table_results_current_year

I extract all the results from the games that had already been played in the current season, from the website resultados-futbol.com.

- Create functions to manage the table

Here I define the functions that I use in order to edit the table with the info I will need in future steps to calculate the predictions. These are the same I used in the 1st notebook.

- Apply functions to table_results_current_year

Time to apply the functions defined above, except for the points_between_teams) which I can only apply when I have all the results from all the years together in one same table.

- Correct Team names of table_results_current_year

Some teams have different names in the table_results_current_year and table_results_previous_years. We need to correct those names so we can fit the model with the right data.

- Join Tables previous years and current year

In this step I join both tables. For that, I read the table_results_previous_years which is saved in my local files, and then I join (concatenate) it with the table_results_current_year which has been created in this notebook in a previous step.

- Apply function points_between_teams

Now that we have one single table with the results of all years, we can apply the remaining function points_between_teams.

- Join the table with fifa data

In this step I join (merge) the table_results with the table_fifa, so we have all the data together in the same dataframe that will be the base to fit the model. First, I load the fifa_table (from my local files) which has been created in the second notebook, and then I use the fuzzywuzzy library to create a dictionary in order to match the names of the teams between the 2 tables. Later, I do some manual corrections to the dictionary and finally I join them. To finish this subsection, I add a column that shows if the match is between 2 rival teams (1) or not (0).

- Train and apply the model

Here, first I extract from the site https://www.resultados-futbol.com/ what are the matches of the current weekday and the following, which are the ones the model will predict. I add a new column to my dataframe called “weekday” that shows if each match is part of the current, following, or other (past) weekday.

Then, I build a dataframe with all the matches that have already finished, and I divide it in X_train (with only the columns that had been used to train the different models in the previous notebook) and y_train. After this, I classify the columns by type (numeric vs categorical), at the same time I include 2 “numeric” columns (year and rivals) as categorical, for the same reason I have explained in notebook 4.

Now I create a new dataframe with only the games I need to predict. I also generate the new variable X_predict, which is another table that includes the data I will use to fit the model. The following step is to create the same pipeline I did before, which applies the transformer StandardScaler to numeric data and OneHotEncoder to the categorical one, and fit the LogisticRegression model with X_train and y_train, using the parameters that showed to be the most accurate of all (C=10). Then, I predict the results (y_pred) from the dataframe X_predict.

- Prepare a dataframe to show the end user the predictions

Here, I add a new column with the predictions (*1 x 2*) and 3 more columns with the probability of each one of those classes. I also add another column with the date in which this notebook is being run (I will use this to show the user in the Streamlit app in which day the predictions were updated for the last time).

- Save the dataframe in the cloud

As a last step, I save the dataframe to a CSV in my local machine, and then I upload it to my Google Cloud Storage account, in a specific bucket I have created for this project. To do this, I had to create first an account service key in my GC account, with the required permissions to access and upload files to the bucket. After creating it, I got a json file with the key I needed, and so I only had to set in the notebook the path where that key is located in my local machine. That’s all I need to upload the CSV to the bucket. It is worth to mention that the specific bucket I created for this project is set to be public to internet. That means the documents I have there can be accessed by anyone with the public URL of the respective file. With this configuration, the PY file that Streamlit runs will not need to use a key to read the file with the predictions.

5.1 Apply Best Model (PY file)

This file is basically a copy of the last notebook described (number 5). It includes all the same sub sections and cells, except for a few lines of code that I have excluded, as it was only to display visual information, such as dataframes. I have done that because we don't really need that, as the objective of it is to have a Python script that when run gives the same final result as the notebook, which is to update the CSV with the predictions on the GCS bucket. It is this file that I use to run automatically with cron (more details about cron in the its respective section of this document).

Streamlit_app_GCS.py

Streamlit defines itself as *an open-source Python library that makes it easy to create and share beautiful, custom web apps for machine learning and data science*. I use this library to build a simple web app in order to show the end user what are the predictions of the football matches, which is the final objective of this whole project.

To build an app with Streamlit, first you need to install the library in your machine. In the page https://docs.streamlit.io/ you have a lot of information about how Streamlit works, and you can also find there instructions to install it.

I decided to create a public app, by creating this python script that reads the CSV with the predictions stored in my bucket in Google Cloud. Everytime the Streamlit app runs, the python script is executed, and the user can see the predictions. Also, I included in the code an option so the user can select what league he wants to see the predictions, and what weekday (if the current or the following). I've also inserted the day of the last update of the predictions CSV, so the user knows when the table was updated for the last time.

Although Streamlit offers the chance of using functions that store its result in a local cache, we don't really need it because the app only needs to read a simple CSV and apply very light modifications to it. Thanks to that simplicity of the PY file, the user can get the results very quickly.

I had also dropped the last update date, country and weekday columns, as they appear above the table. To finish, I had also excluded the index column. I saw in this Streamlit knowledge base article how to do it, and included it in my code.

Using Cron to automate the predictions update

When the user accesses the web app I created with Streamlit, he hopes to see the predictions of the next games. In order for the predictions to be updated, all I need to do is to run the PY file 5.1, which on my Mac takes about 12 minutes. So, I should run it a few times per week, as there are weekdays that happen during the week, and not only on weekends. To avoid me running manually the notebook, I created a cron job on my mac to run that PY file every day, at 11am.

Cron is a standard Unix utility that works as a job scheduling software. It allows to execute commands at specified times or intervals. You can install it in your local machine and schedule emails to be sent, reports to be generated, files to be updated, etc. I use it to run a Python script that updates the predictions table uploaded in my Google Storage Bucket.

So, I configured the following job on my mac:

```
0 11 * * * /opt/anaconda3/envs/tfm/bin/python
/Users/tiago/Master/0.TFM/TFM_Github/5.1.Apply_best_model.py
```

With it, I am basically scheduling my local machine to run every day at 11am (part of the code that is in black), using python (in blue, the route to Python on my Mac), the file 5.1.Apply_best_model.py (in purple, the route to that file on my Mac).

If you want to know more about Cron and how to use it to run notebooks automatically, I recommend you read this article, that was of great help to me: How to Schedule Python Scripts With Cron – The Only Guide You'll Ever Need.

Streamlit Web App for the end user

Now that you know how I build the predictions, is time to see the final result of this project: the public web app I built using Streamlit, that is available in the following URL: https://share.streamlit.io/tiagovilaverde/tfm_kschool/Streamlit_app_GCS.py

There the user will find first the date in which the predictions were updated for the last time. Then, he can select what country (national league) he wants to consult and what weekday: if the current or the following one. After his selection, the table below will show him the matches: what teams are playing, on what day, the result in case any of the games had already been played, and the outcome forecast for all of them, including the probability of each outcome (*1 X 2*).

Conclusion

From the beginning we knew that predicting the outcome of a football match is not easy. There are a lot of factors that can have an impact, and some of them can be quite difficult (if not impossible – like luck) to forecast. Still, by collecting data relative to factors and applying machine learning techniques, it is possible to make predictions with a certain level of accuracy.

In this project, the factors I have used to predict were the year, country, home and away teams, rivalry between them, their ratings, fatigue, season and recent form, and game's history. I collected data from the year 2005 to 2021 (17 years in total) to predict results of the season 2022. For that I trained different models with different estimators and parameters, and the best one resulted to be the Logistic Regression with an F1 score a little above 0.47.

The F1 score we got is much higher than the 0.27 we get if we apply a Dummy Classifier. However, it can certainly be improved. If I had to work on a better model, I would start by feeding it more data, that include factors that were not part of this project. Examples: information relative to players, how many goals each team has scored and conceded, the name of the coach of each team, the weather during the match, the referee, etc. Also, trying other algorithms and parameters combination could also contribute for a more reliable forecast.

In the end, it is clear that machine learning can help us to make better predictions, in a wide range of fields. Training the right model with the right data can lead to solid forecasts which would be difficult to get otherwise, even for some fields with high levels of uncertainty such as the outcome of a football match.