Université Côte d'Azur

Master Informatique, parcours Intelligence Artificielle

Project report

Football Predictor

Authors :
BOULLI Marouan, BAPTISTA Rafael and HADDOU Amine

Supervised by :
FORMETI Enrico

Due Date :
4 décembre 2023

Football Predictor

CouscousSamba

19/11/2023

Table des matières

1 Project

The ”Football Predictor” project is undertaken as part of the Neural Network course, under the guidance of Professor [Insert Professor’s Name]. The primary objective is to develop an artificial intelligence model capable of predicting the outcomes of football matches, specifically identifying the winning team. This endeavor leverages two datasets, which will be detailed in the subsequent sections, to train and optimize the predictive model.

2 Project Presentation

This project aims to utilize machine learning techniques to predict football match outcomes, blending the realms of sports analytics and artificial intelligence. The foundation for our predictions is laid upon two datasets - the ”International football results from 1872 to 2023” and the ”FIFA World Ranking 1992-2023.” The former comprises over 45,000 international football match results, spanning from the inaugural match in 1872 to 2023, sourced from various international football associations. The latter dataset contains FIFA rankings for men’s national teams from 1992 to (July) 2023. These datasets, referred to as ”results” and ”ranking,” respectively, serve as the building blocks for our predictive model.

In the end, we should be able to predict match results and the winner of a championship.

To that end, we followed 3 steps :

1. **Data preprocessing**.
2. **Models training and evaluation**.
3. **User interface development**.

The source code is available at the following github repository : Football match result predictor

2.1 Members

The collaborative effort behind the Football Predictor project is led by the following team members :

- Marouan Boulli
- Raphael Breaud
- Amine Haddou

Each member brings a unique set of skills and expertise to the project, fostering a diverse and capable team.

2.2 Datasets

The project relies on two key datasets obtained from Kaggle's website :

1. ”International football results from 1872 to 2023,” available for download here.
2. ”FIFA World Ranking 1992-2023,” accessible here.

The first dataset encompasses a comprehensive collection of over 45,000 international football match results, spanning from the inception of official matches in 1872 to 2023, sourced from diverse international football associations. The second dataset provides FIFA rankings for men's national teams from 1992 to (July) 2023. These datasets, denoted as ”results” and ”ranking,” form the cornerstone for our predictive modeling efforts.

3 Organisation

3.1 GitHub

To ensure efficient collaboration and version control, we employ GitHub as a central repository for our project. While most work is conducted locally using notebooks (due to Git merge challenges with notebooks), regular updates are pushed by a designated team member to maintain the GitHub repository's currency.

3.2 Kanban Board

Project management is facilitated through a Kanban board, which serves as the organizational backbone for our team. Tasks are assigned to each team member, providing a comprehensive overview of the project's progress and enabling individuals to clearly understand their responsibilities.

3.3 Discord

Discord acts as the primary communication tool for our team. We have established a dedicated server where regular video conferences are held, fostering real-time communication and collaboration. The platform is also utilized for sharing ideas and progress updates, providing a space for constructive debates on various algorithms and methodologies.

4 Preprocessing

In this section, we delve into the preprocessing steps applied to the datasets.

4.1 Fixing Country Names

The dataset posed challenges with inconsistent country names, including countries that no longer exist, those that underwent name changes, and variations in syntax. To address this, we employed two methods :

1) Utilizing the following ChatGPT prompt among some others that are included in this one

ChatGPT Prompt:

> Instructions: You are a prompt engineer specializing in modern geography and country names, as well as football (soccer). Your task is to process a list of countries provided in various languages. Some of these countries may have specific conditions or requirements that need to be addressed. Here are the instructions to be followed: 1. You will receive a list of countries in different languages. 2. Some of these countries may no longer exist, have different names, appear multiple times in different formats, be in a non-simplified form (e.g., IR Iran instead of Iran), or not have a national football team. 3. These countries will be referred to as "false countries." 4. Your output should be in JSON format, where each key represents a "false country," and its corresponding value indicates the correct actual name in English. 5. If a "false country" doesn't have a national football team, its value in the JSON should be "None." Please ensure that your response strictly adheres to the provided instructions and only includes the requested JSON output.

we generated a dictionary to replace problematic country names with correct ones (e.g., URSS -> Russia, IR IRAN -> IRAN, Reunion -> France).

2) Mapping countries in the datasets to a Python library (insert library name) that contains the official names of all countries. This process homogenized the names using abbreviations, and we dropped countries for which we lacked information.

4.2 Choosing a Year Cut-off

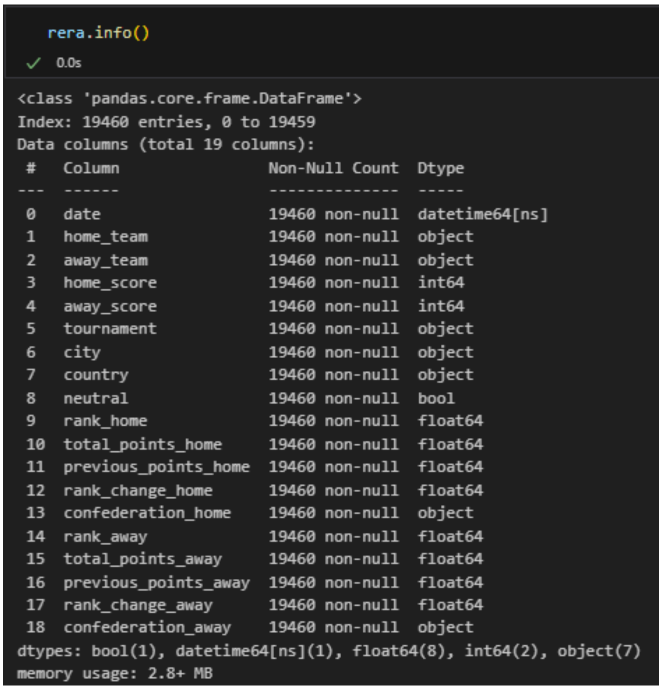
After thorough consideration, we opted to establish a ”cut-off year” based on the project's performance rather than alterations in country names. Data prior to the year 2000 is deemed largely irrelevant due to the following reasons :

- Matches from that era have minimal impact on current game outcomes due to their age.
- Significant changes in game dynamics and player preparation over the years render comparisons between past and present matches less meaningful.
- The structure of competitions has undergone substantial transformations, making team performances from the past less indicative of their current capabilities.

This initial cut-off year allows us to filter out less relevant historical data. Subsequently, we plan to fine-tune the cut-off year further to enhance the precision of our predictive model.

4.3 Features

In the process of constructing our predictive model, not all features in the dataset prove to be relevant. As such, a careful selection of the most meaningful features becomes



```
rera.info()
✓ 0.0s
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 19460 entries, 0 to 19459
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   date                  19460 non-null  datetime64[ns]
 1   home_team             19460 non-null  object
 2   away_team             19460 non-null  object
 3   home_score            19460 non-null  int64
 4   away_score            19460 non-null  int64
 5   tournament            19460 non-null  object
 6   city                  19460 non-null  object
 7   country               19460 non-null  object
 8   neutral               19460 non-null  bool
 9   rank_home             19460 non-null  float64
 10  total_points_home     19460 non-null  float64
 11  previous_points_home  19460 non-null  float64
 12  rank_change_home      19460 non-null  float64
 13  confederation_home    19460 non-null  object
 14  rank_away             19460 non-null  float64
 15  total_points_away     19460 non-null  float64
 16  previous_points_away  19460 non-null  float64
 17  rank_change_away      19460 non-null  float64
 18  confederation_away    19460 non-null  object
dtypes: bool(1), datetime64[ns](1), float64(8), int64(2), object(7)
memory usage: 2.8+ MB
```

FIGURE 1 – List of features before the modifications

imperative. While there is no universally perfect solution for feature selection, our approach involves leveraging our soccer expertise as fans to make informed choices.

Traditionally, machine learning tasks often require expert input for feature selection. In our case, we consider ourselves soccer enthusiasts with sufficient knowledge to make informed decisions regarding feature relevance.

Here is list of features before the modifications at 1.

4.3.1 New Features

In our quest for optimal feature selection, we introduce new features to enhance the predictive capabilities of our model.

Winner Feature : The 'winner' feature is introduced as an integer column with three distinct values : {0, 1, 2}.

- **0 :** Indicates that the *home_team* has emerged victorious.
- **1 :** Indicates that the *away_team* has secured a win.
- **2 :** Represents a draw, signifying that both teams have achieved an equal outcome.

Average Goal Feature : We incorporate two new features, *home_goals_avg* and *away_goals_avg*, representing the average number of goals scored by the home and away teams, respectively, in their last 7 matches. This choice is grounded in the notion that observing the goal-scoring patterns over the most recent seven matches provides valuable insights into a team's current performance.

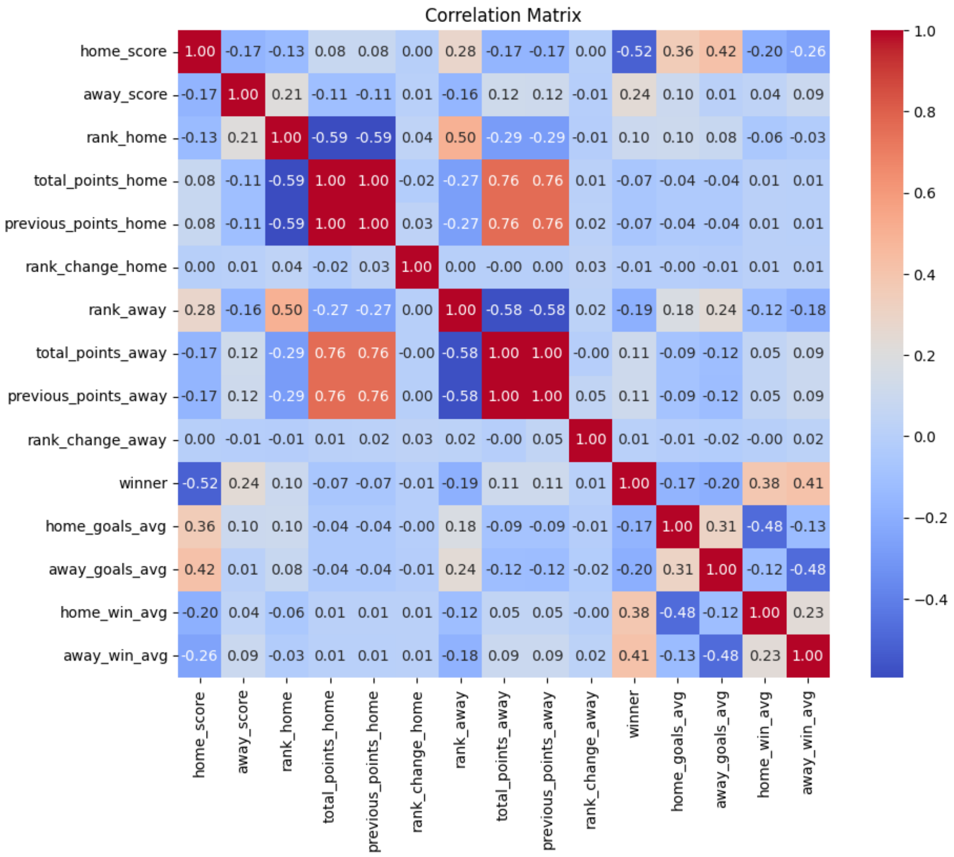


FIGURE 2 – The correlation heatmap of the dataset

Average Win Feature : To further enrich our feature set, we introduce two additional features, *home_win_avg* and *away_win_avg*. These columns represent the average number of victories for the home and away teams in their last 7 matches. This metric offers a snapshot of a team's recent success rate, helping to capture its current form over a reasonably recent timeframe.

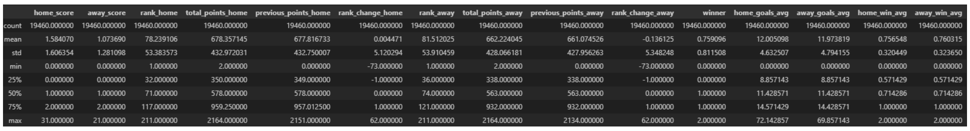
Last Five Confrontations : Finally, we find it valuable to compute the number of victories for each team in their last five matches. This addition is motivated by the common observation in soccer, where a team may have a higher statistical standing than its opponent but still faces a likelihood of losing. Such occurrences can be attributed to intangible factors like pressure, intensity, and the overall context of the game.

4.3.2 Deleted Features

In our pursuit of a more streamlined and effective model, we have opted to eliminate certain features to mitigate the risk of overfitting and enhance the interpretability of our results.

To identify the features slated for removal, we employ a combination of intuitive assessment of seemingly irrelevant features and mathematical insights 3. One crucial metric guiding our decisions is the correlation coefficient. The computed correlations on our dataset are presented in the figure at 2.

City feature : The feature is representing the location where a match is held, is dee-



	home_score	away_score	rank_home	total_points_home	previous_points_home	rank_change_home	rank_away	total_points_away	previous_points_away	rank_change_away	winner	home_goals_avg	away_goals_avg	home_win_avg	away_win_avg
count	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000	19460.000000
mean	1.584070	1.073690	78.239106	678.357145	677.816733	0.004471	81.512025	662.224045	661.074526	-0.136125	0.759096	12.005098	11.973819	0.756548	0.760315
std	1.606354	1.281098	53.383573	432.972031	432.750007	5.120294	53.910459	428.066181	427.956263	5.348248	0.811508	4.632507	4.794155	0.320449	0.323650
min	0.000000	0.000000	1.000000	2.000000	0.000000	-73.000000	1.000000	2.000000	0.000000	-73.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.000000	0.000000	32.000000	350.000000	349.000000	-1.000000	36.000000	338.000000	338.000000	-1.000000	0.000000	8.857143	8.857143	0.571429	0.571429
50%	1.000000	1.000000	71.000000	578.000000	578.000000	0.000000	74.000000	563.000000	563.000000	0.000000	1.000000	11.428571	11.428571	0.714286	0.714286
75%	2.000000	2.000000	117.000000	959.250000	957.012500	1.000000	121.000000	932.000000	932.000000	1.000000	1.000000	14.571429	14.428571	1.000000	1.000000
max	31.000000	21.000000	211.000000	2164.000000	2151.000000	62.000000	211.000000	2164.000000	2134.000000	62.000000	2.000000	72.142857	69.857143	2.000000	2.000000

Figure 3 – Basic statistics of numerical features

med irrelevant to our analysis given the focus on national teams. Consequently, we have decided to drop this feature. The city where a match is played is considered inconsequential, as the outcome of national team matches is influenced more significantly by the country in which they are contested.

Country feature : In a similar vein, we have chosen to discard the *country* column. The presence of a *neutral* column renders this redundant. If the match is not set as neutral (neutral = false), it is held in the home country. If it is, it is played in another country worldwide. The distinction between specific countries does not significantly impact the analysis in this context.

home_score and away_score features : Furthermore, we will exclude the *home_score* and *away_score* features since we already possess the *winner*, *home_goals_avg*, and *away_goals_avg* features. The introduction of average goal features provides a more nuanced perspective on team performance, considering that the number of goals alone may not fully capture a team's overall effectiveness. By removing the score columns, we aim to enhance the model's generalization capability and reduce the risk of overfitting. Anyway, the scores should not be present in the features since the column ”winner” (label) has been created using the scores.

previous_points_home and previous_points_away Features : As depicted in the correlation heat map 2, *previous_points_home* and *previous_points_away* are respectively highly correlated with *total_points_home* and *total_points_away.* Over multiple matches, the number of points does not vary significantly, potentially explaining this correlation. We posit that the total number of points alone suffices, rendering the information from *previous_points* redundant, despite its slight variations.

4.3.3 Standardization

Standardization is a crucial preprocessing step in machine learning that aims to bring numerical features to a common scale, ensuring that they contribute equally to the model's training process.

In our context, standardizing the numerical data involves transforming each feature such that it has a mean of 0 and a standard deviation of 1. This not only removes potential biases introduced by different measurement units but also facilitates a more stable and efficient convergence during the training phase.

By standardizing the numerical features, we create a more uniform and fair representation of their influence on the model, promoting a robust and unbiased learning process.

4.3.4 Encoding String and Boolean Type Features

In our effort to optimize the model's performance, we considered encoding country names to address the challenges posed by String-type data. For instance, Germany might be represented as 0, England as 1, and France as 2.

Two primary encoding methods, Hotcoding and encoding, were evaluated. While the former initially seemed more logical, it resulted in an unwieldy number of new columns, unnecessarily complicating the model.

As a result, we decided on encoding. The columns selected for encoding include : *home_team*, *away_team*, *confederation_home*, *confederation_away*, and *neutral.*

However, subsequent research revealed that for Principal Component Analysis (PCA) to be efficient, it is advisable to drop all categorical features, as PCA relies on the variation of numerical data. Consequently, encoding is unnecessary in our particular case.

4.3.5 Dimensionality Reduction

Dimensionality reduction is a technique used to decrease the number of features in a dataset while retaining crucial information. This approach serves various purposes, including reducing model complexity, enhancing the performance of learning algorithms, and facilitating data visualization.

To implement dimensionality reduction, we will employ the Principal Component Analysis (PCA) algorithm. Specifically, we are applying PCA to our encoded dataframe, `rera_standardized_encoded`, to determine the optimal value for the hyperparameter `n_component`. This identified value will subsequently be incorporated into the model notebook's pipeline, streamlining the automated encoding and application of dimensionality reduction.

To choose the best hyperparameter, we need to select the abscissa where the curve reaches its maximum.

In our context, as mentioned earlier, we chose the second case where we neither encoded nor used the categorical features 5, as our research has convinced us of its appropriateness.

4.3.6 Fine-tune winning feature

The objective of our model is to predict the winners of various matches during a tournament. In the current state of our dataset, the model is configured to predict one of the following outcomes : a win for the home country, a loss for the home country (equivalent to a win for the away country), or a neutral result.

However, for our specific use case, we only want to predict wins or losses (not neutral outcomes) because, in a tournament format, matches cannot end in a neutral result.

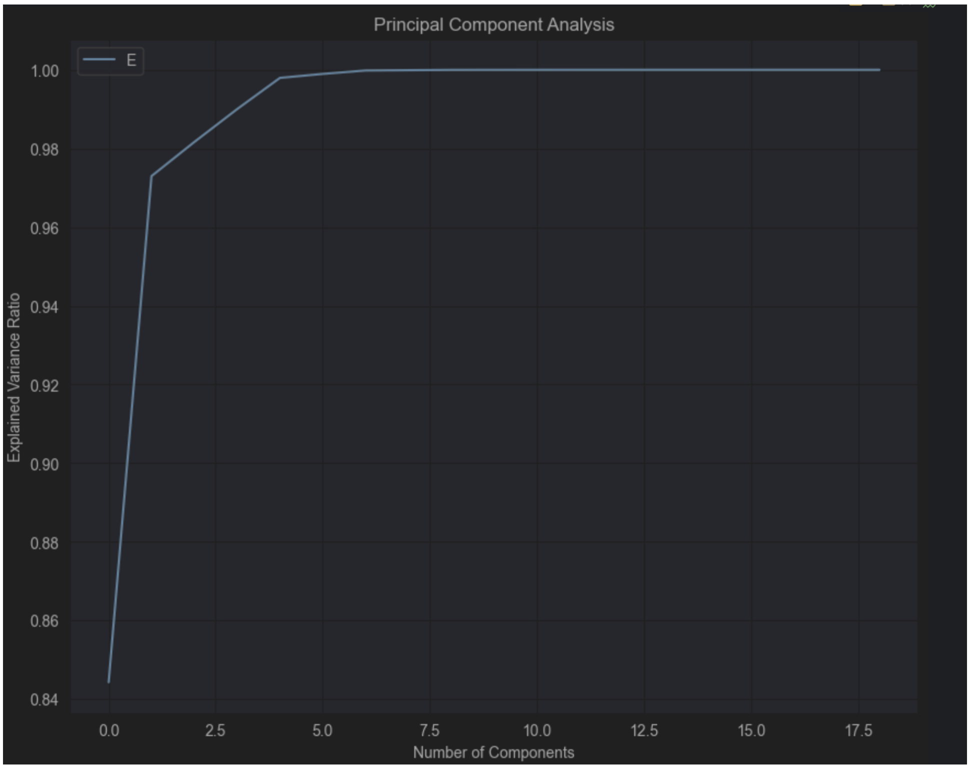


FIGURE 4 – PCA's hyper parameter with encoding

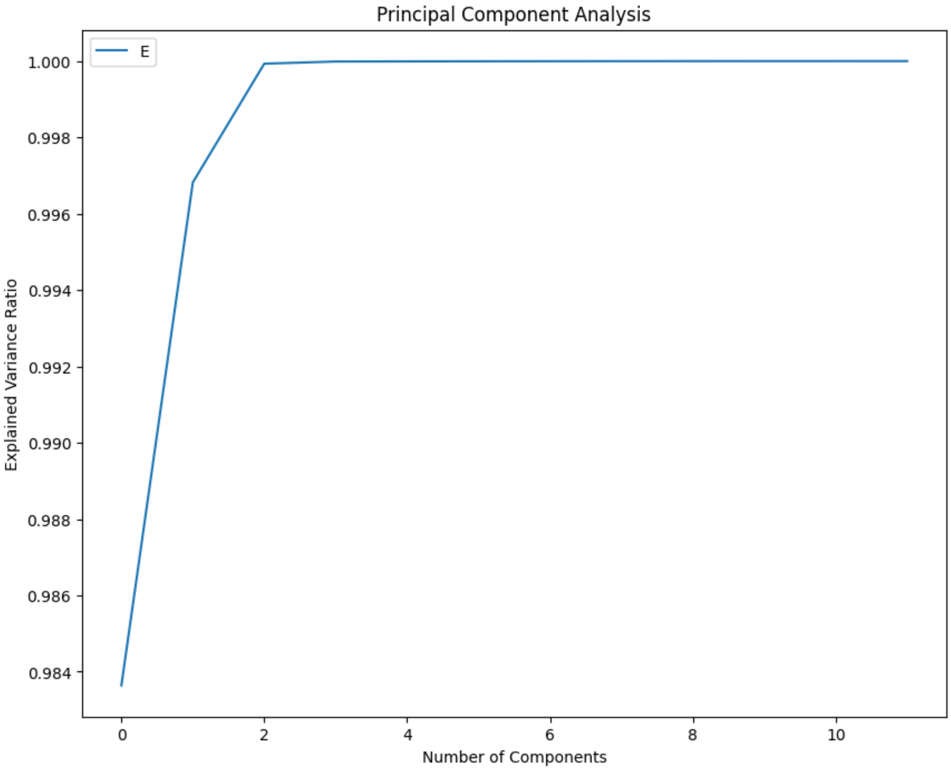


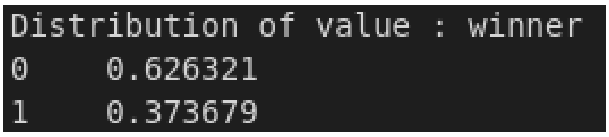
FIGURE 5 – PCA's hyper parameter without encoding

Initially, we had 4628 neutral matches. We removed all neutral matches for friendly games, resulting in 2774 remaining neutral competitive matches. We then utilized data from ”shootouts.csv” to determine the winners of these matches where possible, reducing the number to 2561 (mainly from qualifiers where shootouts were not applicable).

Considering the relatively small size of the neutral dataset and the absence of alternative solutions, we made the decision to exclude all neutral matches from our dataset.

4.3.7 Imbalanced data

Analysis reveal that classes are imbalanced :

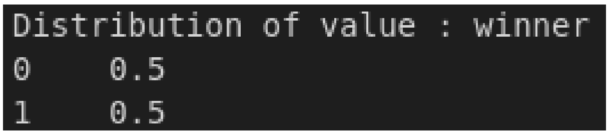


```
Distribution of value : winner
0    0.626321
1    0.373679
```

Figure 6 – Imbalanced classes

This may introduce bias to the model which could struggle at predicting correctly the minority class.

To tackle this issue, a random oversampling is done on the minority class, duplicating random data from the minority class. This way, we get balanced classes.



```
Distribution of value : winner
0    0.5
1    0.5
```

Figure 7 – Balanced classes

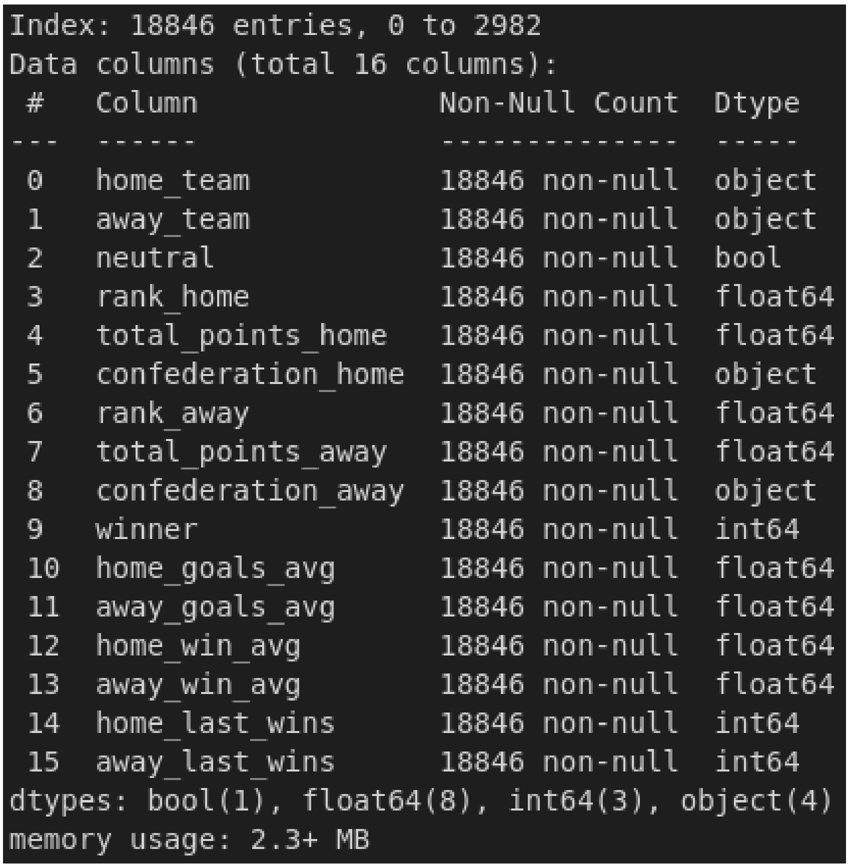
However, as we will mention later, it may introduce overfitting as some data is redundant.

Nonetheless, there are more advanced techniques that are worth a try like SMOTE to deal with imbalanced data.

4.4 Summary

With the completion of all modifications and adjustments, our dataset is now prepared for further analysis. We have taken necessary steps to mitigate biases and address the curse of dimensionality. The dataset is now primed for training machine learning models.

Here’s an overview of the features of the dataset preprocessed.



```
Index: 18846 entries, 0 to 2982
Data columns (total 16 columns):
 #   Column              Non-Null Count  Dtype
---  ------              --------------  -----
 0   home_team           18846 non-null  object
 1   away_team           18846 non-null  object
 2   neutral             18846 non-null  bool
 3   rank_home           18846 non-null  float64
 4   total_points_home   18846 non-null  float64
 5   confederation_home  18846 non-null  object
 6   rank_away           18846 non-null  float64
 7   total_points_away   18846 non-null  float64
 8   confederation_away  18846 non-null  object
 9   winner              18846 non-null  int64
 10  home_goals_avg      18846 non-null  float64
 11  away_goals_avg      18846 non-null  float64
 12  home_win_avg        18846 non-null  float64
 13  away_win_avg        18846 non-null  float64
 14  home_last_wins      18846 non-null  int64
 15  away_last_wins      18846 non-null  int64
dtypes: bool(1), float64(8), int64(3), object(4)
memory usage: 2.3+ MB
```

Figure 8 – Preprocessed dataset features

5 Models

In this section, we will analyze the different models used to predict match results and the results obtained.
We train and evaluate three different kinds of models :

- Random Forest
- Multi Layer Perceptron Classifier (MLPC)
- Logistic regression

All models are trained and evaluated using default hyperparameters **AND** optimized hyperparameters (Grid search). The data used ("rera.joblib") has been preprocessed in the preprocessing notebook, as described before.
In the end, the goal is to choose the best model to predict match results during the competition.

Before training the model, features must be preprocessed :

- Encode categorical features using one hot encoding method : models only deal with numbers, so we must convert non numbers features to numbers.
- Standardize numerical features : to avoid introducing bias related to different scales across columns. If a column has numbers much greater than other columns, the model may give more importance to this column, whereas it shouldn't.

If we want to feed the model with new data for prediction, this data must be preprocessed the same way as during training.

So, in order to anticipate and simplify the prediction phase, we used scikit-learn pipelines to include this preprocessing step during training and prediction.

Pipelines are objects that allow to define several steps and apply fit() and transform() methods on each of these steps sequentially. Each step must implement fit() and transform() methods. In our case, the pipeline has 2 steps :

- Preprocessing : contains standardization and one hot encoding steps.
- Classifier : contains the model.

This way, we can call the fit and predict methods on the Pipeline object, and both preprocessing and training or prediction are automatically executed, so we don't have to do the preprocessing manually each time we want to predict on new data.

The Pipeline 9 object is then exported to joblib format, and we can use it in any program we want, without having to preprocess data again.

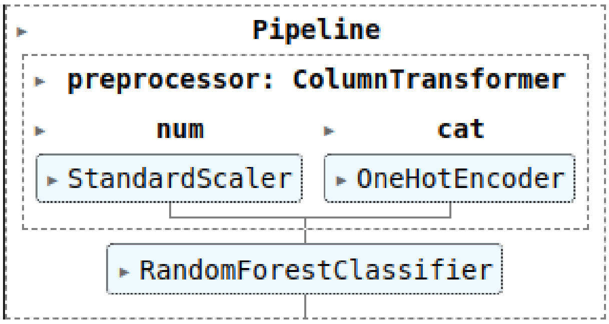


FIGURE 9 – Pipeline

Note : for all the experiments, data is split the same way, i.e a third is dedicated to evaluation and two thirds for training. Models predict 2 classes : 0 (home team wins) and 1 (away team wins). A draw is predicted when predicted probabilities range between 0.45 and 0.55.

5.1 Random Forest

The first model trained is Random Forest.

Random Forest is an ensemble learning algorithm used for classification and regression in machine learning. It constructs multiple decision trees during training, where each tree is trained on a random subset of the data and considers a random subset of features at each decision point. The model outputs the mode of classes for classification or the mean prediction for regression based on individual tree outputs. This randomness and diversity help reduce overfitting, enhancing overall predictive performance. The final prediction is obtained by aggregating the predictions of all trees, resulting in a robust and versatile model known for high accuracy and generalization across diverse datasets.

5.1.1 Default hyper-parameters

The model is first trained using default hyper-parameters to get a baseline evaluation. Random state value is fixed to 42 to be able to reproduce the results.

Here's the confusion matrix obtained :

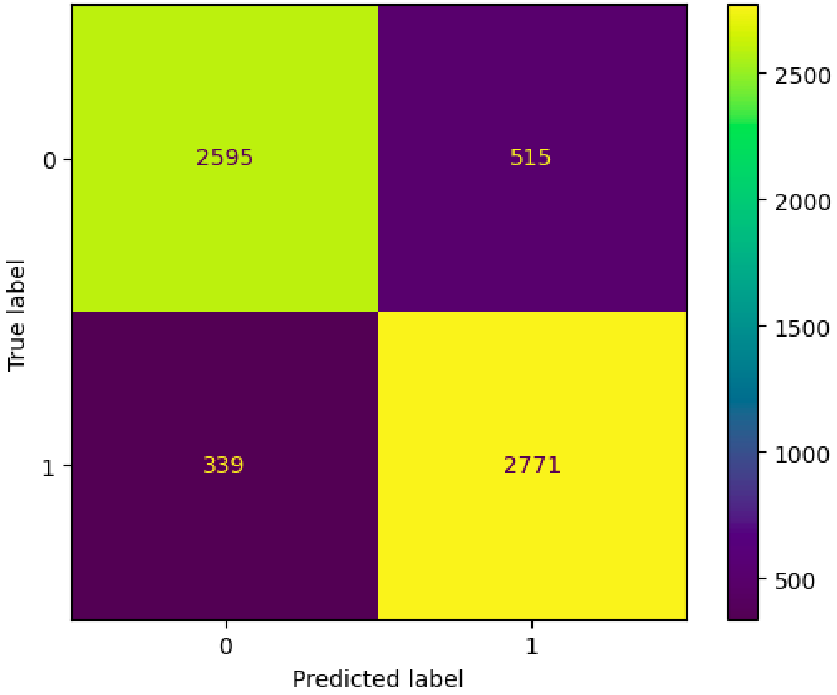
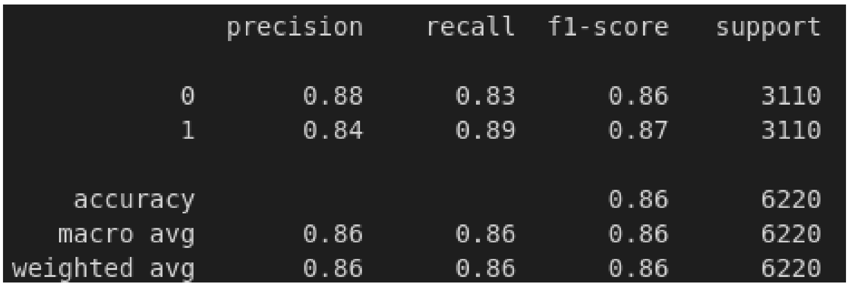


FIGURE 10 – Confusion matrix rf default

The confusion matrix reveals that the model is a little bit better at predicting the class ”1” corresponding to away team wins. This observation applies to all training configuration, and may be attributed to the oversampling done to get balanced classes (the minority class over-sampled is the class ”1”).

Evaluation metrics are reported in the classification report.



```
              precision    recall  f1-score   support

           0       0.88      0.83      0.86      3110
           1       0.84      0.89      0.87      3110

    accuracy                           0.86      6220
   macro avg       0.86      0.86      0.86      6220
weighted avg       0.86      0.86      0.86      6220
```

FIGURE 11 – Classification report rf default

The accuracy score is pretty good : 0.86. The model has a good representation of the data and is able to extract interesting information from the features in order to predict the result of a match.

Nonetheless, learning curves 12 shows a not negligible gap between the validation and the training curves, indicating that the model is perhaps overfitting the data, and therefore may struggle to generalize to new data. (it's not sure because we don't see the curve going down)

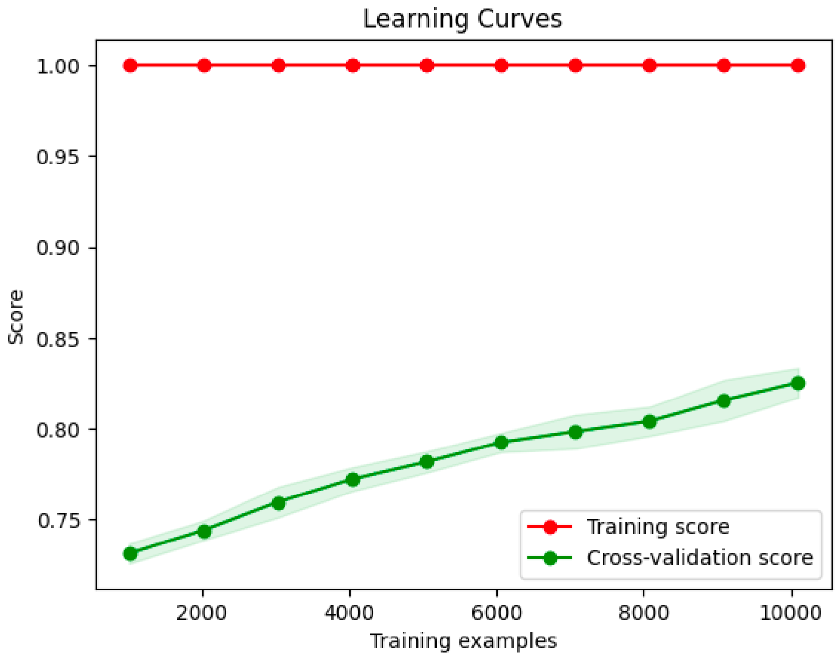


FIGURE 12 – Learning curves rf default

It may be due to the random oversampling used to balance classes. By duplicating data, we must have influenced the model to overfit.

It's also possible that it's due to the fact that the Random Forest is too complex to represent the data we're using.

As a consequence, it's likely that the model won't be able to generalize very well, even if the accuracy is high.

To address this issue, there are some avenues to explore :

- Use another method for oversampling to balance the classes (like SMOTE algorithm).
- Further fine-tune the model hyperparameters.
- Use a simpler model like Logistic Regression.

5.1.2 Fine-tuned hyper-parameters

Model hyperparameters are the configuration settings used to structure and guide the learning process of a machine learning model. These hyperparameters are set before the training process and play an important role in model's performance improvement and overfitting/underfitting control.

Some examples of hyperparameters for Random Forest model are :

- **n estimators** : refers to the number of trees in the forest. Generally, more trees increase the performance and make the predictions more stable, but they also make the computation slower.

- **max depth** : sets the maximum depth of each tree. Deeper trees can model more complex patterns but might lead to overfitting. Conversely, shallower trees might be too simplistic (underfitting).

- **max features** : determines the number of features to consider when looking for the best split during the construction of the trees.

There are other hyperparameters, but these ones seem to be a good starting point to optimize model's performance.

An exhaustive search to find the best configurations for these hyperparameters is not possible. Thus, it's required to define the hyperparameters search space :

- **n estimators** : [50, 100, 500].

- **max depth** : [10, 50, 100].

- **max features** : [None, 'sqrt', 'log2'].

Using Grid Search CV from scikit-learn library, we can find the best hyperparameters combination in the defined search space.

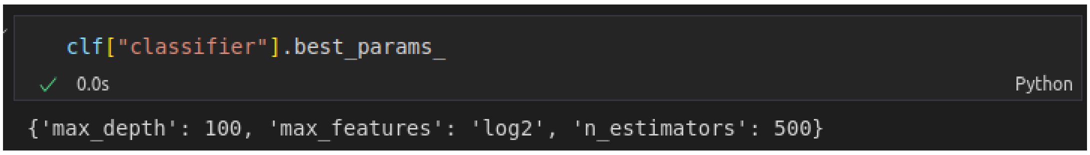
For each combination of parameters in the search space, Grid Search CV performs K-fold cross-validation. This means the data is split into K subsets (folds) ; the model is trained on K-1 of these folds and validated on the remaining one. This process is repeated K times, with each fold being used once as the validation set. Cross-validation helps in assessing the model's performance more robustly, reducing the risk of overfitting on the training set.

After training on each combination, Grid Search CV evaluates the model's performance using a scoring metric, like accuracy.

Once all combinations have been evaluated, Grid Search CV selects the combination of parameters that yielded the best performance according to the chosen metric.

Finally, the model is retrained on the entire dataset using the best-found parameters.

Best hyperparameters values found :



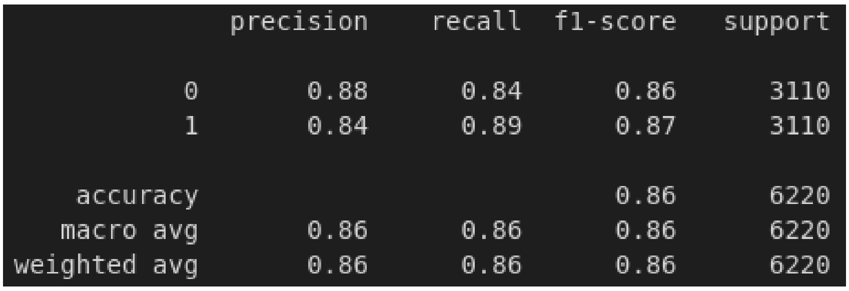
```python
clf["classifier"].best_params_
```

```
{'max_depth': 100, 'max_features': 'log2', 'n_estimators': 500}
```

FIGURE 13 – Best hyperparameters rf

Unfortunately, the hyperparameters fine-tuning did not improve model's accuracy 14, nor did it help reduce the gap between training and validation curves 15.

Further experiments with different search spaces should be done to improve the results.



```
              precision    recall  f1-score   support

           0       0.88      0.84      0.86      3110
           1       0.84      0.89      0.87      3110

    accuracy                           0.86      6220
   macro avg       0.86      0.86      0.86      6220
weighted avg       0.86      0.86      0.86      6220
```

FIGURE 14 – Classification report rf gridsearch

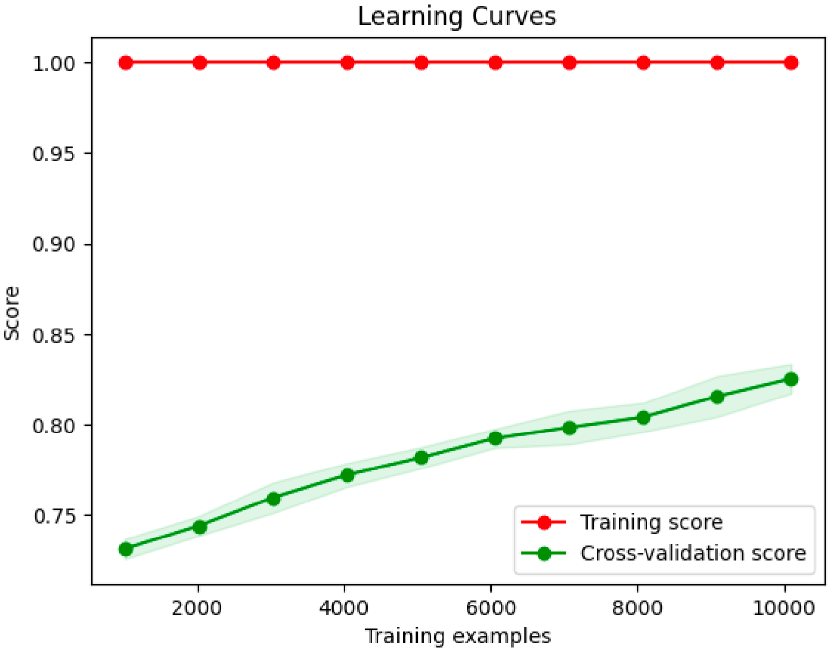


FIGURE 15 – Learning curves rf gridsearch

5.2 Multi-Layer Perceptron Classifier (MLPC)

The Multi-Layer Perceptron Classifier (in scikit-learn) is a type of neural network model used for classification tasks. It belongs to the family of feedforward artificial neural networks and is used to solve problems that require supervised learning.

MLPC consists of at least three layers of nodes : an input layer, one or more hidden layers, and an output layer. Each node in the hidden layers is a neuron that uses a non-linear activation function, typically ReLU (Rectified Linear Unit).

Weights are adjusted trying to minimize the **cross entropy** (log loss function) :

$$H(y, \hat{y}) = -\sum_{i} y_i \log(\hat{y}_i) \tag{1}$$

This loss is a representation of the difference between predicted values and true values. By minimising this function, we improve the model's accuracy.

By default, the MLPC of scikit-learn uses the Adam (Adaptive Moment Estimation) optimizer to update weights minimising the loss function. This optimizer is well-known for its efficiency, as it allows to adjust the learning rate for each parameter of the model, making it adaptive and faster than other algorithms to converge to the minima of the loss function.

5.2.1 Default hyperparameters

Like Random Forest, the MLPC model is first trained using default hyper-parameters to get a baseline evaluation. Random state value is fixed to 42 to be able to reproduce the results.

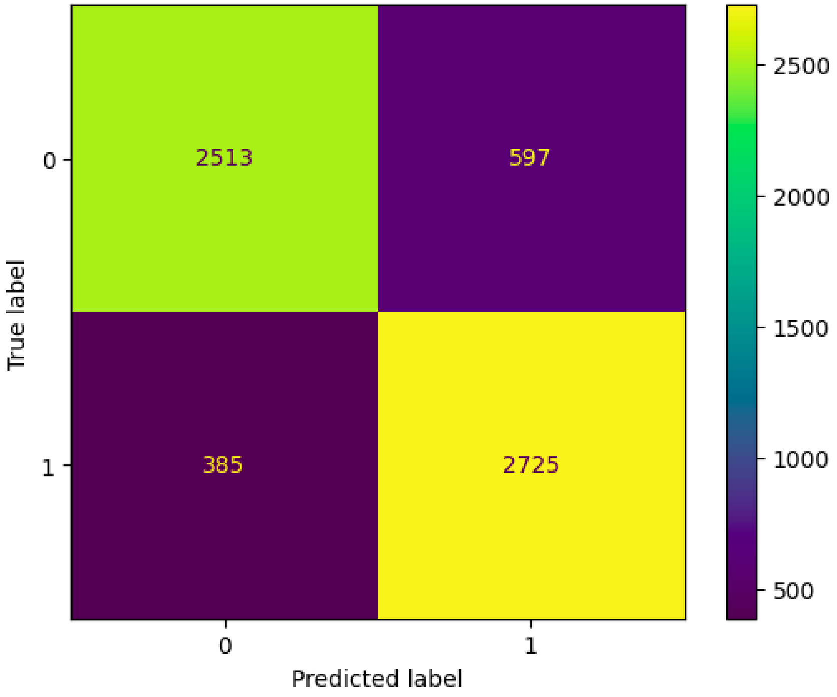
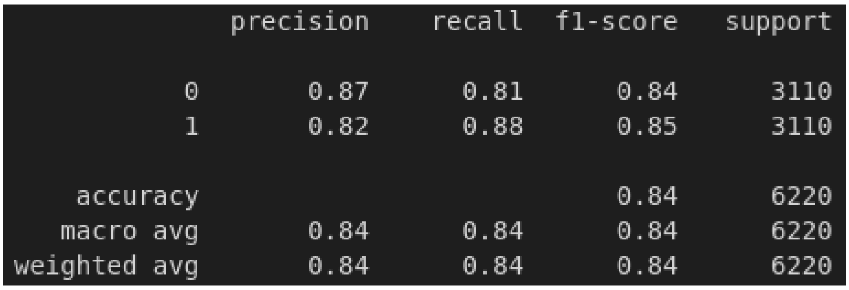


FIGURE 16 – Confusion matrix mlpc default

Same observations as for Random Forest are made : the model is better at predicting the class ”1” (when the away team wins) than ”0” (when the home team wins).

Classification metrics results are reported in the classification report :



```
              precision    recall  f1-score   support

           0       0.87      0.81      0.84      3110
           1       0.82      0.88      0.85      3110

    accuracy                           0.84      6220
   macro avg       0.84      0.84      0.84      6220
weighted avg       0.84      0.84      0.84      6220
```

FIGURE 17 – Classification report mlpc default

Results are a little bit worse than Random Forest (0.84 < 0.86), but still good.

However, like for Random Forest, learning curves suggest that the model may be overfitting :

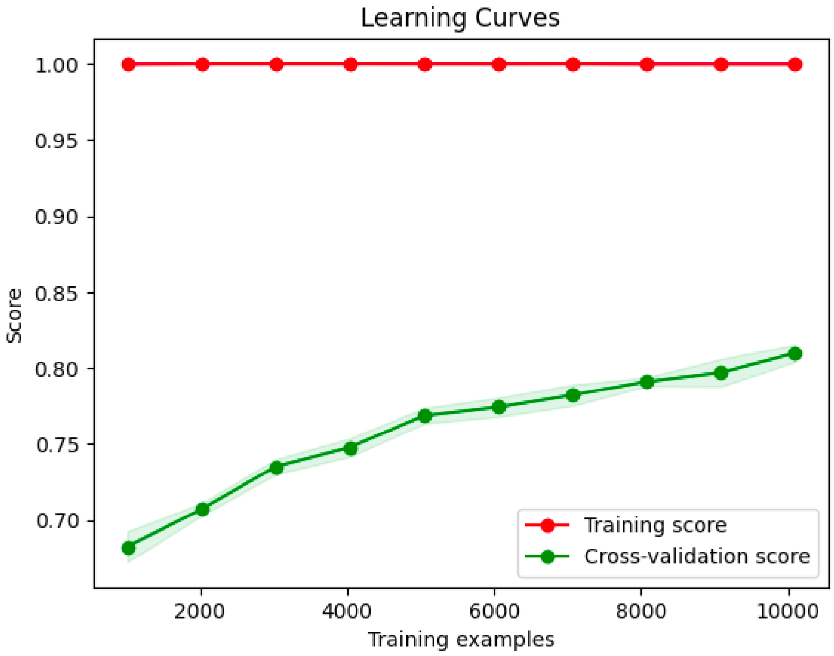


FIGURE 18 – Learning curves mlpc default

Let's try to fine-tune the hyperparameters to improve the accuracy and reduce the gap between learning curves.

5.2.2 Fine-tuned hyperparameters

Like for Random Forest model, there are a lot of hyperparameters. For convenience and gain of time, we chose the following two hyperparameters to optimize :

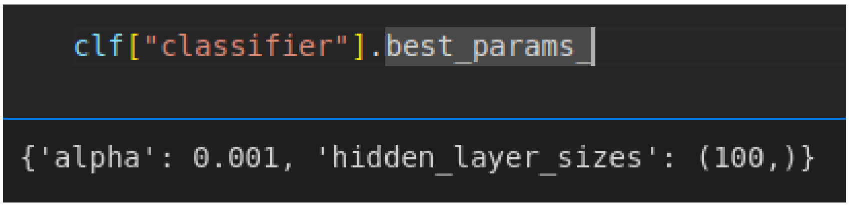
- **hidden layer sizes** : determines the size and number of hidden layers in the neural network. Overfitting often occurs when a model is too complex relative to the amount and complexity of the training data. By fine-tuning hidden layer sizes, we can control the model's complexity.
- **alpha** : L2 regularization term (also known as Ridge regularization) helps to avoid overfitting by penalizing large weights. Regularization adds a penalty to the loss function for complex models, encouraging simpler models that generalize better. A higher value of alpha increases the regularization effect, which means the optimizer prefers simpler models with smaller weights. This can prevent the model from fitting the training data too closely.

As said before, an exhaustive search is not possible and the time of computation required to run the grid search increases a lot when the search space is greater. We have to restrict to a relatively small search space (although we used njobs parameter to parallelize computations) :

- **hidden layer sizes** : [(50,), (100,), (50, 50)]

- **alpha** : [0.0001, 0.001, 0.01]

Best hyperparameters values found :

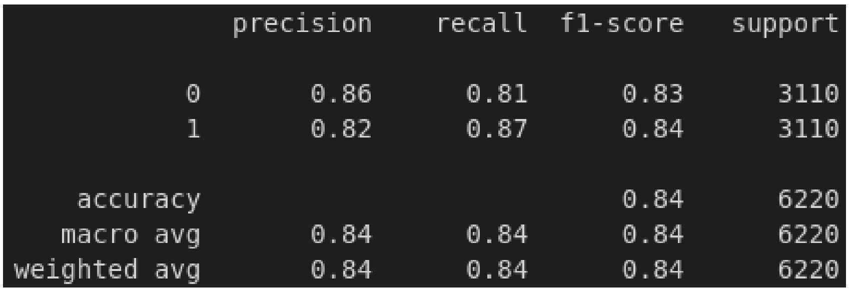


```
clf["classifier"].best_params_
```

```
{'alpha': 0.001, 'hidden_layer_sizes': (100,)}
```

FIGURE 19 – Best hyperparameters values mlpc

Like for Random Forest, the hyperparameters fine-tuning did not improve model's accuracy 20, nor did it help reduce the overfitting 21.



```
              precision    recall  f1-score   support

           0       0.86      0.81      0.83      3110
           1       0.82      0.87      0.84      3110

    accuracy                           0.84      6220
   macro avg       0.84      0.84      0.84      6220
weighted avg       0.84      0.84      0.84      6220
```

FIGURE 20 – Classification report mlpc gridsearch

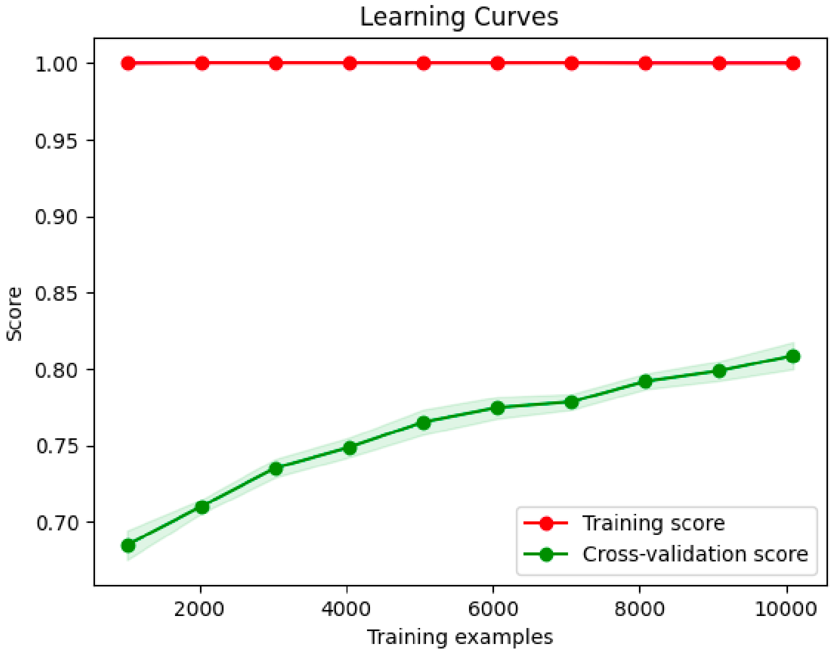


FIGURE 21 – Learning curves mlpc gridsearch

5.3 Logistic regression

Logistic regression is a statistical method used for binary classification tasks. It's a type of regression analysis that's suitable for predicting the probability of a binary outcome (typically with two possible outcomes, like 'win' or 'loose', 'success' or 'failure', etc.).

It models the probability that a given input belongs to a particular category. It does this by using a logistic function to map predicted values to probabilities. This function outputs a value between 0 and 1, which can be interpreted as the probability of the instance belonging to a particular class.

Being a simpler model than Random Forest and MLPC, one can assume that it will not overfit the data, but at the risk of underfitting it.

Let's find out.

Here's the confusion matrix :

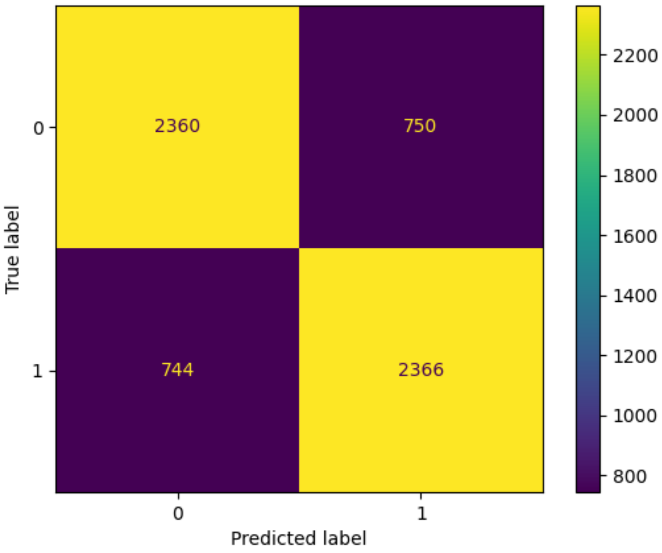
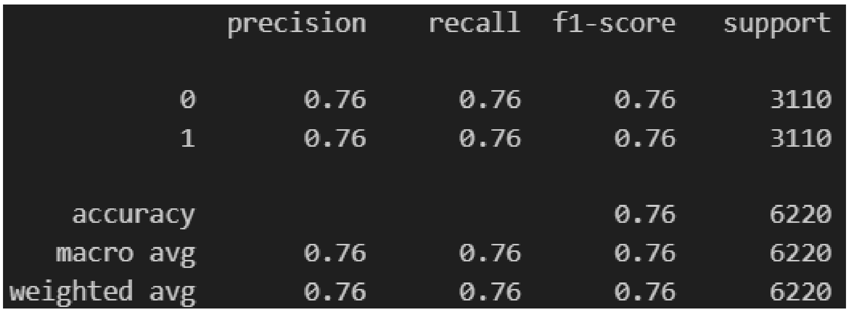


FIGURE 22 – Confusion matrix logistic regression

And the classification report :



```
              precision    recall  f1-score   support

           0       0.76      0.76      0.76      3110
           1       0.76      0.76      0.76      3110

    accuracy                           0.76      6220
   macro avg       0.76      0.76      0.76      6220
weighted avg       0.76      0.76      0.76      6220
```

FIGURE 23 – Classification report logistic regression

We observe that scores are lower than Random Forest and MLPC.

Let's have a look at the learning curves to check if the model is potentially overfitting or underfitting.

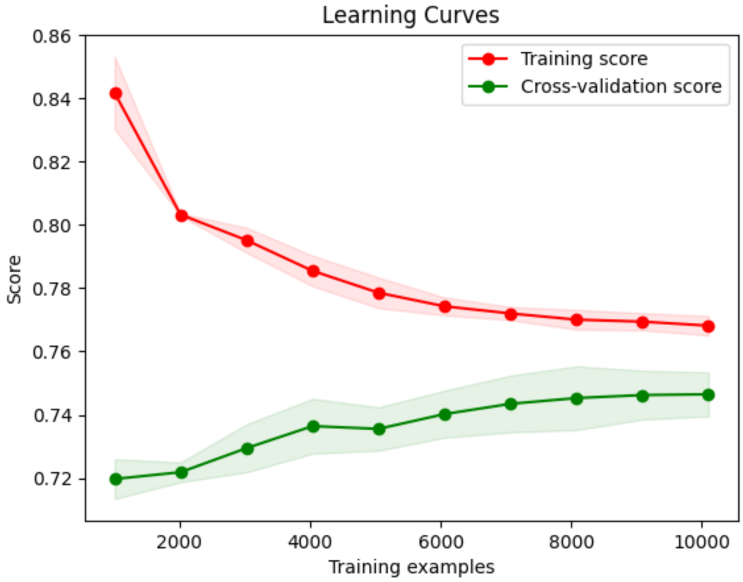


Figure 24 – Learning curves logistic regression

The gap between the curves is smaller, we're no more suspecting an overfitting, but the scores are also lower. Given the previous learning curves, we know that the training score can be really high, thus one can suspect an underfitting here, suggesting that the model is not good enough to correctly represent the data.

5.4 Summary

Although accuracy scores are pretty good, Random Forest and MLPC models have a non negligible gap between the training and the validation curves, which suggests a potential overfitting.

As a consequence, the model may not be able to generalize well.

As for the Logistic regression model, learning curves are close to each other, but scores are lower, suggesting a potential underfitting.

Hyperparameters fine-tuning did not help reduce the gap between learning curves, nor did it improve the accuracy. Further experiments with different search spaces should be done to improve the results.

As we're not sure about overfitting, we choose the model with the highest accuracy : Random Forest with optimized hyperparameters.

Note : MLPC model size is much smaller than Random Forest (1MB < < 49MB). But since memory is not really a constraint in our project, we still choose Random Forest.

6 Python-Based User Interface for Predictive Models

To elegantly expose and allow users to test our models, we have implemented a Python-based user interface using Flask. The interface, hosted locally as a website, leverages the Flask library for seamless interaction.

6.1 User Interface Structure

The user interface comprises three distinct HTML pages : ”Welcome,” ”Predict Match,” and ”Predict Tournament.” Each page serves a unique purpose to enhance user experience.

- **Welcome Page :** This initial page greets the user and provides the option to choose between the two prediction modes.
- **Predict Match Page :** Users can utilize this page to forecast match results.
- **Predict Tournament Page :** Designed in the FIFA World Cup format, this page allows users to set up a tournament and predict the eventual winner.

6.2 Key Features

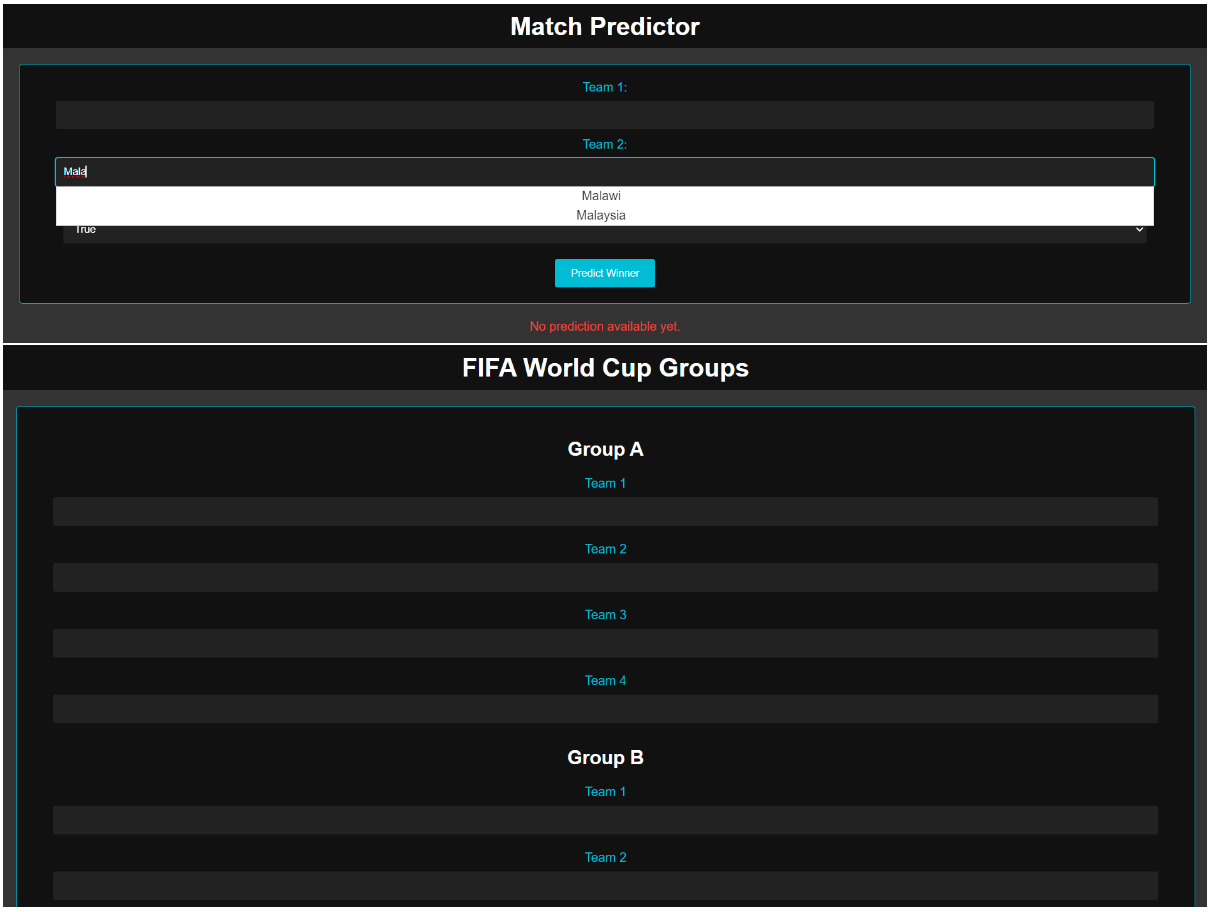
The user interface incorporates several key features to improve usability and provide valuable insights :

- **Auto-completion for Country Names :** To prevent typos and save time, the interface includes auto-completion functionality for country names.
- **Probability Display :** The interface dynamically displays the probability of a winner, conveying the likelihood of a team’s success.

6.3 Application Structure

The entire application, along with all necessary files, is organized in the **app** directory. This directory contains the following subdirectories :

- **assets :** Contains the models used in the user interface.
- **data :** Contains the data used in the file, specifically for the tournament.
- **static :** Contains the CSS file.
- **templates :** Contains the UI templates, encompassing the welcome page, match predictor page, and tournament predictor template.
- **app.py :** The main function driving the user interface.
- **predict_match_result.py :** Module containing functions to predict match results.
- **predict_championship.py :** Module with functions to predict tournament winners.



No prediction available yet.

FIGURE 25 – Snaphshots of the UI

7 Conclusion

The main purpose of this project is to be able to predict the result of football matches and the winner team of a championship.
To that end, scripts and a user interface allowing users to predict matches and championship results are provided.

We first preprocess the data in order to have relevant features compatible with the model input by :

- Selecting a cutting year and standardizing country names to fit with current Fifa countries.
- Removing unrelevant features.
- Adding new relevant features.
- Standardizing numerical features and encoding the categorical ones to fit with the model constraints.

Then, we train several models with different configurations, evaluate them and choose the best model regarding the accuracy metric. Globally, accuracies are pretty good, but there's a large gap between the training and the validation curves for the Random Forest and the MLPC models, and a potential underfitting for the Logistic regression model. The performed analysis leads us to choose the Random Forest model.

Finally, we provide python scripts to predict the match result between 2 teams and the winner of a championship. We also provide a user interface to test these functionalities in a more user friendly way.

This project is really interesting since it covers almost all parts of a machine learning project, from data preprocessing to model deployment.

The data preprocessing part seems to be the most challenging one since the model relies on the features coming from this data preprocessing, and it's not always easy to find the best way to clean and select relevant data to maximise the model performance.

Some improvements can be made to this project :

- Try to preprocess data using different cutting years and compare the model performances.
- Find other relevant features to fit to the model.
- Compare more models with deeper hyperparameters optimization.
- Improve the user interface ergonomics.