ELEN0062 - Introduction to machine learning

Pass prediction during football matches

Prof. Wehenkel Louis - Prof. Geurts Pierre
TA: Antonio Sutera

Group 3

Debor Antoine	Antoine.Debor@student.uliege.be
Deflandre Guilian	G.Deflandre@student.uliege.be
Navez Pierre	Pierre.Navez@student.uliege.be

Academic year 2020-2021

Contents

1 Introduction

As part of the course ELEN0062-1 - Introduction to machine learning, this project has been set up aiming to link what has been studied in the theoretical course and a common problem of life. The goal of this project is to design a model able to predict the next player who will receive the ball via a pass during a football game and this based on the position of all players and the ball at a given time.

Sports analysis has been a subject of constant evolution over the past decade, in professional sports leagues around the world.

Sports Analytics has been a steadily growing and rapidly evolving area over the last decade both in US professional sports leagues and in European football leagues. In view of the recent implementation of strict financial fair play regulations in European football but also of the current craze for sports betting, Sports Analytics will surely be a very important topic in the years to come. The majority of techniques used in the field so far are statistical. However, there has been growing interest in the Machine Learning and Data Mining community about this topic.[5]

This report clarifies the methodology adopted by our group in order to tackle the problem and present an efficient solution.

2 Data

For the successful completion of this project, data has been provided divided into a training set and a test set. The input data have the following fields

- `ID`: the id of the pass made during a game;
- `time_start`: the number of milliseconds since the beginning of the concerned half-time period;
- `sender`: the ID of the player who has the ball (in $[1, 22]$);
- `x_<ID>`: the x position of the player `<ID>` (in $[-5250\ (cm); +5250\ (cm)]$);
- `y_<ID>`: the y position of the player `<ID>` (in $[-3400\ (cm); +3400\ (cm)]$);

The output corresponding to this input on the other hand contains the field

- `receiver`: the ID of the player who receive the ball via a pass (in $[1, 22]$).

2.1 Data set analysis

First, an analysis of the raw data provided has been done in order to have prior knowledge about the dynamics of the passes during a game.

The percentage of successful passes has been computed. A pass is considered as successful when both players between the pass are in the same team. The resulting analysis indicates that 82,8% were successful, and we thus decided to consider failed ones in our learning process as they represent a non negligible part of the data. Moreover, none of them came back to the sender.

In addition, the mean distance of a pass over all the passes in the learning samples is calculated. This mean distance is 15.88 meters and the standard deviation associated is 9.33 meters. The longest pass is 70.20 meters long and the shortest is 0 meters long. These information provide a region of interest around the sender of the ball.

2.2 Data preprocessing

2.2.1 Pairs of players

In order to solve the problem, the same approach than the one suggested in the `toy_example.py` file has been followed. For a given pass in the learning set, 22 pairs of players are derived. A pair of player consists in a tuple containing the IDs of the sender and a potential receiver. For the sake of simplicity, the pair (sender, sender) has been kept even if it should not lead to successful passes.

In order to solve the problem given a set of position and some IDs, it is mandatory to capture the implicit information contained in a pass. These information should indicate the model what is the current state of the game. It can be related to the sender environment, to the potential receiver environment, to the feasibility of a pass between the sender and a potential receiver, or even to the game state in general such as an attacking phase. Thus, for each pair of players, some features have been added. They are explained in the next section.

2.2.2 Quantitative Features

It has been decided to enrich our data set with some numerical features, mainly being distances over the football field. Let's review which features have been selected to solve the prediction problem.

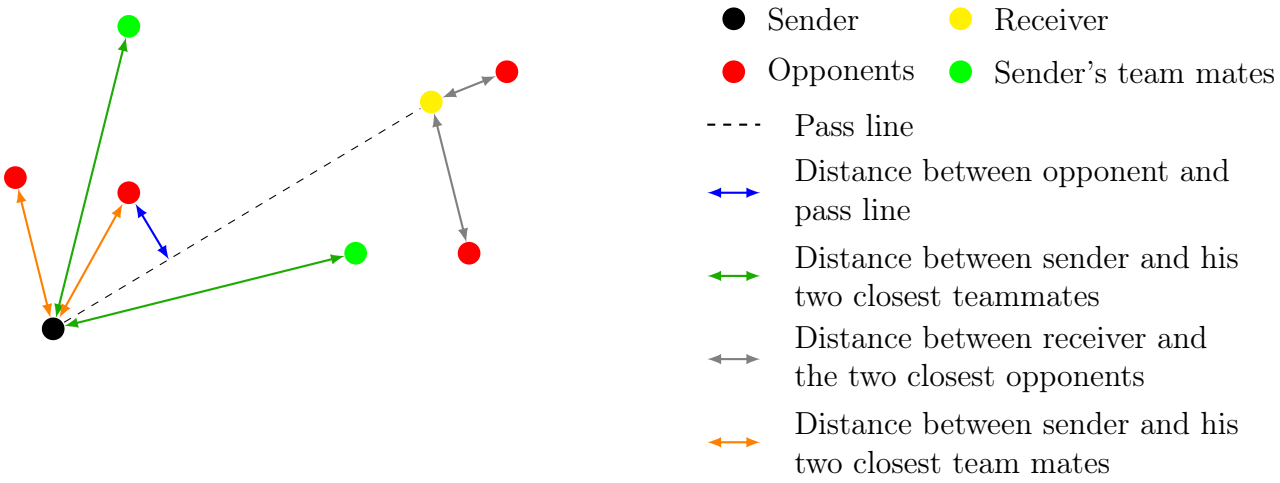


FIGURE 2.1: Computed distances

2.2.2.1 Distance between the sender and the receiver

This feature was already implement by the `toy_example.py` provided file and has been kept. It simply consists of the Euclidean distance[1] between the ball sender and the potential receiver (drawn among all players on the field including the sender). In FIGURE 2.1, this is show by the length of the pass line.

2.2.2.2 Distances between the sender and his two closest team mates

These features are represented in FIGURE 2.1 by the green arrow and correspond to the distance between the sender of the pass and his two closest teammates.

[1]Considering the locations $I = (x_i, y_i)$, $J = (x_j, y_j)$ over the football field (***i.e.*** $x \in [-5250 \text{ cm}; 5250 \text{ cm}]$, $y \in [-3400 \text{ cm}; 3400 \text{ cm}]$), the Euclidean distance is given by

$$d(I, J) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2}$$

2.2.2.3 Distances between the receiver and his two closest opponents

These features has also been added and are the Euclidean distances between the receiver of the ball during the pass and his two closest opponents around him. These distances are shown in FIGURE 2.1 in gray for a pass between two members of the same team.

2.2.2.4 Distances between the sender and his two closest opponents

These features has also been added and are the Euclidean distances between the sender of the ball during the pass and his two closest opponents around him. These distances are shown in FIGURE 2.1 in orange for a pass between two members of the same team.

2.2.2.5 Distance between the pass line and its closest sender's opponent

An important feature for the team members was the smallest distance between a member of the sender's opposed team and the pass line. Indeed, if some opponent are between the sender and the receiver, it seems logical that the pass is more risky and therefore that the sender will tend to choose another receiver.

At first, to compute such a distance, two formulas have been used. The first one is Heron's formula. Knowing the distance between the sender S and the receiver R, this formula allowed to compute the area of each triangles made by this pair and an opponent in the field O_i as

$$\mathcal{A} = \sqrt{p\,(p - |SR|)\,(p - |RO_i|)\,(p - |O_iS|)}, \quad p = \frac{|SR| + |RO_i| + |O_iS|}{2}$$

Once the area of all these triangles computed, the general triangles area's

$$\mathcal{A} = \frac{\text{base} \times \text{height}}{2}$$

can be used in order to get back the smallest distance between O_i which is basically the height of the triangle SRO_i. At first, the minimal height between all the one computed (one for each opponent of the sender) has been thus chosen to get the wanted feature.

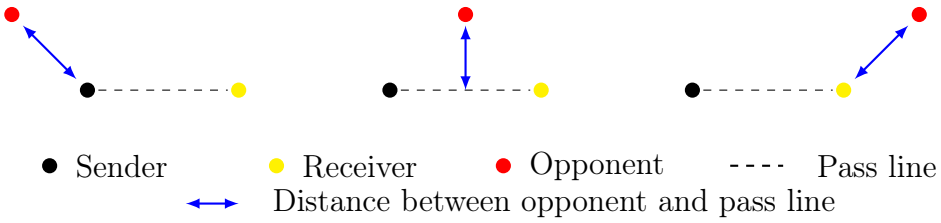


FIGURE 2.2: Shortest distance of point to a line segment.

The problem was that using these formulas provided us the shortest distance of an opponent to the line RS, but not to the segment $[RS]$. Indeed, when one computes the shortest distance of a point to a line segment, three scenarios can happen which are depicted in FIGURE 2.2. When calculating the distance of a point (here the opponent O_i) to a line segment (here the pass line, ***i.e.*** the line between the sender S and the receiver R), the first thing to find is if the projection of this point O_i on the line SR belongs to the segment $[SR]$. If it's the case, then the shortest distance can be computed as the height of SRO_i as explained before. If it's not, then the shortest distance is the minimal value of the distances between the point and the segment extremities (***i.e.*** $\min\{|O_iS|, |O_iR|\}$).[4]

Finally, this feature has thus been upgraded to compute the distance between the opponent and the pass segment, rather than the pass line. This is illustrated by the blue double arrow in FIGURE 2.1.

2.2.2.6 Distance between the sender/receiver and the field center

These features correspond to the Euclidean distances between the sender and the receiver of the ball during the potential pass and the center of the football field located in $(0, 0)$.

2.2.2.7 Gain on the abscissa of the ball

This feature represents the horizontal movement of the ball towards the goal of the team opposed to the one of the sender.

To compute this attribute , the side of the field occupied by each team has to be known. To determine this, the algorithm begins by determining the leftmost player on the field, considering that it is the goalkeeper of one of the 2 teams. If this player is on the same team than the ball sender, the goal in which he will have to score is on the right of the field, otherwise it is on the left.

Knowing that, it is therefore trivial to compute the horizontal displacement of the ball (Δx)) for each potential pass made by the sender. Formally, for a given pass from the sender at (x_s, y_s) and a receiver located at (x_r, y_r), we have

$$\Delta x = x_r - x_s$$

if the sender team's goal is on the left side, and

$$\Delta x = -(x_r - x_s)$$

if it is on the right side.

Note that here, an abstraction is made: *a player will never be located behind the goalkeeper of the team on the left side of the field.* It has been assumed that these situations are rare in football and thus that this is a right way to guess teams' sides.

2.2.2.8 Ordinate displacement of the ball

This new data has been added to the model by simply computing the absolute value of the difference of ordinates coordinates between the sender and the potential receiver of a pass. Formally, for a given potential pass from the sender at (x_s, y_s) and a receiver located at (x_r, y_r), the ordinate displacement (in absolute value) Δy is given by

$$\Delta y = |y_r - y_s| .$$

2.2.3 Qualitative Features

2.2.3.1 Predicate indicating if the sender is on the same team as the receiver of not

This feature was already implement by the `toy_example.py` provided file and has been kept. It simply consists of a boolean value being 1 (***i.e.*** `True`) if the potential receiver of a pass is a team member of the sender, and 0 (***i.e.*** `False`) otherwise.

2.2.3.2 Predicate indicating if the sender's team is performing an attack or not

In a football game, the pass performed can depend of whether the team possessing the ball performs an attack or not. It thus has been decided to implement a function able to determine if the sender is in an offensive or defensive phase of game. To do this, first the sender's team field side is determined using the same method as in section 2.2.2.7. After that, the mean value of the x location of each player on the sender's team is computed, ***i.e.***

$$\tilde{x} = \frac{\sum_{x=a}^{b} x_i}{11}$$

where $[a, b] = [1, 11]$ or $[a, b] = [12, 22]$ depending of the sender's team, i representing the index of a player and 11 being the number of players in a team. Using this value and the side of the sender's

team on the field, we can determine where most of the players in a team are located. If they are mostly in their camp, the action is considered as a defensive one, if they are not it's an offensive one. For instance, if the sender play in the left side of the field (***i.e.*** $x \in [-5250, 0]$) and $\tilde{x} > 0$, it's an attack phase.

2.2.3.3 Number of sender's opponents around the receiver

This characteristic simply gives the number of opponents in a radius of 15,8 meters from the ball receiver. This distance has not been chosen by chance. Indeed, as already mentioned in section 2.1, it represents the mean length of a pass for the whole data set.

2.2.3.4 Location zone of the sender and the receiver on the field

As it can be seen in FIGURE 2.3, it has been decided to split the field into several zones. Indeed, each of us watched football games and we found that the pass habits were not similar depending on which part of the field passes were made. Hence, the following zones definition was decided

- **Zone 1 (center):** $x \in [-3200 \text{ cm} ; 3200 \text{ cm}]$, $y \in [-1750 \text{ cm} ; 1750 \text{ cm}]$;
- **Zone 2 (left goal):** $x \in [-5250 \text{ cm} ; -3200 \text{ cm}]$, $y \in [-1750 \text{ cm} ; 1750 \text{ cm}]$;
- **Zone 3 (right goal):** $x \in [3200 \text{ cm} ; 5250 \text{ cm}]$, $y \in [-1750 \text{ cm} ; 1750 \text{ cm}]$;
- **Zone 4 (high flank):** $x \in [-5250 \text{ cm} ; 5250 \text{ cm}]$, $y \in [1750 \text{ cm} ; 3400 \text{ cm}]$;
- **Zone 5 (low flank):** $x \in [-5250 \text{ cm} ; 5250 \text{ cm}]$, $y \in [-3400 \text{ cm} ; -1750 \text{ cm}]$.

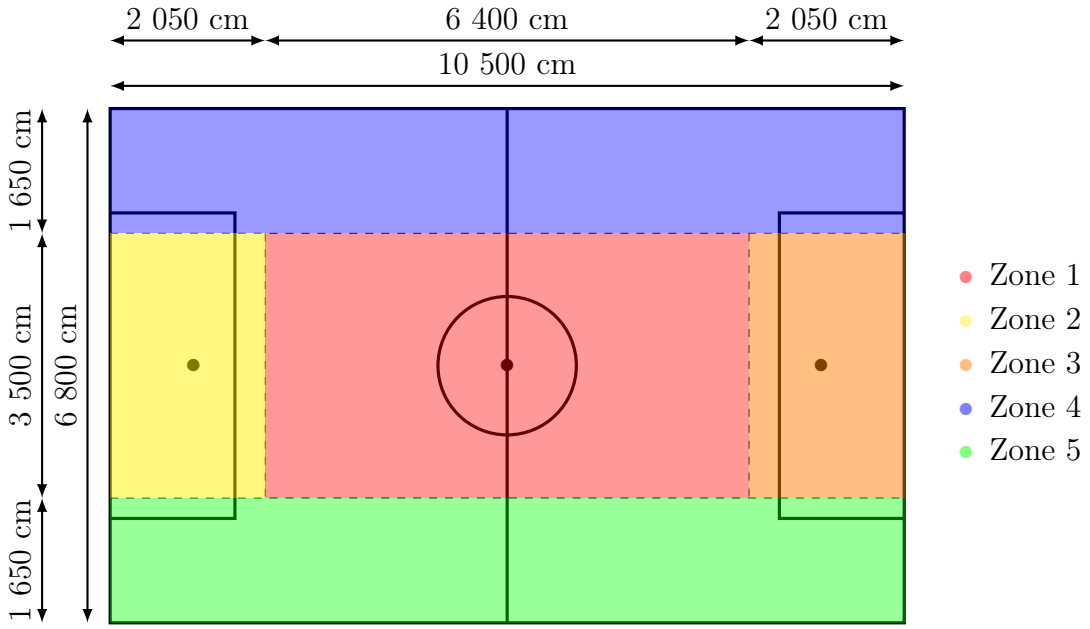


FIGURE 2.3: Football field division

Note that it has been though that both location zones from the sender and the receiver was interesting features. Indeed, a player from the central zone can be more interested to send the ball into one of the lateral zone or one of the goal zones, depending on the other features.

3 Model selection and performance estimate

Once the aforementioned features computed, a model selection has to be performed in order to design the best classifier. This process consists of trying different combinations of features alongside with different learning algorithms (and for each of these testing different hyper-parameters settings) in order to choose the combination providing the best results in term of prediction over a test set derived from the original data set.

3.1 Method

To compare different algorithms and their hyper-parameters, we performed the so-called test set method, which divides the learning data set into three parts, namely the learning set LS, the validation set VS and the test set TS. In the scope of the considered problem, this method consists of the following steps

1. We first randomly split the original 8682 passes data set into a 5210 passes LS and 1736 passes VS and TS, corresponding to 60 and 20 % of the whole data set respectively.
2. Considering a certain combination of features, we then train the set of candidate models on LS (each object being a pair of players, as previously depict).
3. We evaluate each of these on VS by predicting the probability for each pair of players of VS to correspond to a pass, by deriving from those probabilities the predicted receiver's Id for each pass, and by comparing these results with the true output values corresponding to VS. Performing this evaluation allows us to select the best model based on its performance on VS.
4. We then train the selected model on $LS + VS$, and evaluate its performance on TS using the `Scikit-Learn` metric `accuracy_score`, which gives a performance evaluate of the considered model.
5. We finally train the selected model on $LS + VS + TS$ to reach the best accuracy.

3.2 Results

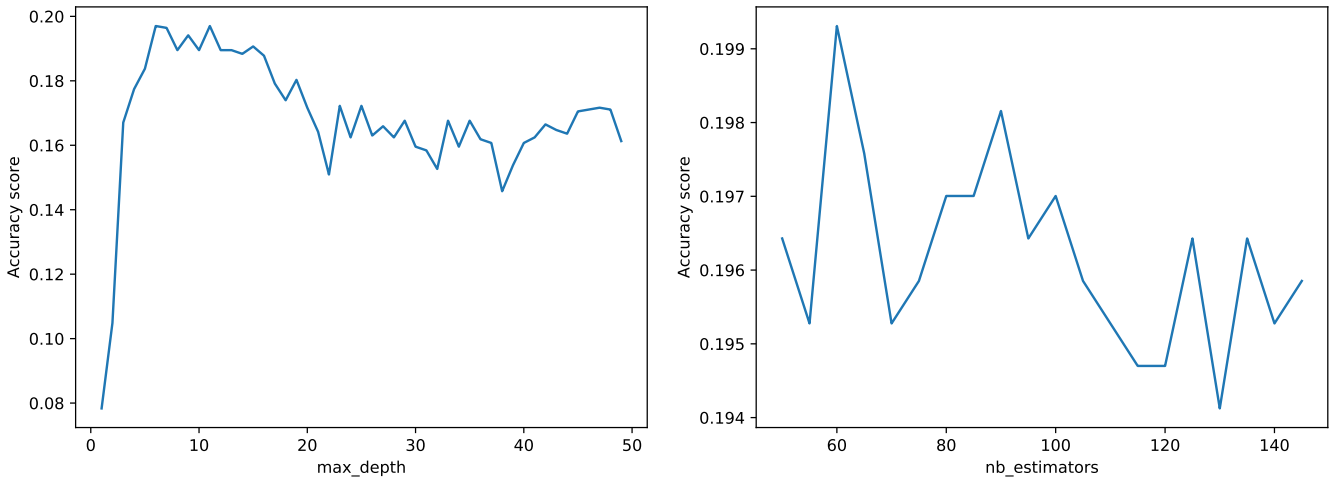
3.2.1 First trials

At first, we only thought about features 2.2.3.1, 2.2.2.1, 2.2.3.3, 2.2.2.5, 2.2.3.4 and the distance between the potential receiver and his two closest sender's opponents. Using these and following the previously described test method, we performed several tests, trying different learning algorithms from the `Scikit-Learn` Python library: a random forest classifier, a k-nearest neighbors classifier, an *Adaboost* classifier and a multi-layer perceptron neural net.

Random forest classifier

For this classifier, we decided to test different values of the hyper-parameters `max_depth` and `n_estimators`, corresponding to the maximal depth of the trees used by the algorithm and to the amount of those trees, respectively.

We hence generated the *accuracy score(maximal depth)* curve of FIGURE 3.1a and the *accuracy score(number of estimators)* curve of FIGURE 3.1b.



(a) maximal depth candidates (b) number of estimators candidates

FIGURE 3.1: Random forest classifier : model selection curves

The first test provided a best maximal depth equal to 6, giving a predicted performance score of 19.20%. The second one has been generated for this fixed maximal depth, and provides a best `n_estimators` equal to 60 and a predicted score of 18.61%. However, we can see in FIGURE 3.1b that the accuracy range is quite narrow, and we can deduce that the value of `n_estimators` has no big impact on the accuracy of the model.

KNN classifier

For this classifier, we decided to test different values of the hyper-parameter `n_neighbors` only[2], corresponding to the number of neighbors considered by the KNN algorithm.

We hence generated the *accuracy score(number of neighbors)* curve of FIGURE 3.2.

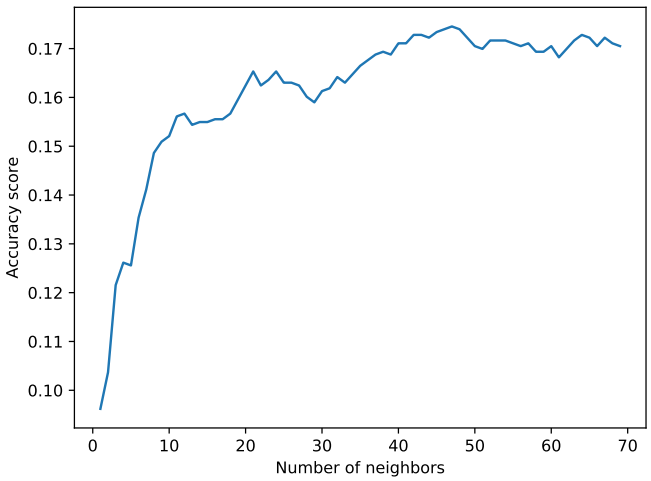


FIGURE 3.2: KNN classifier : `n_neighbors` selection curve

[2]We also considered the `weights` parameter, which can be set to *'uniform'* (by default) or *'distance'* in order to weights the neighbors regarding their distance to the considered point. Since weighting did not provide better results, we gave up this idea.

The test set method provided a best number of neighbors value equal to 47, giving a predicted score of 18.32%. However, we can see in FIGURE 3.2 that the accuracy curve reaches kind of a plateau and we expect models with a `n_neighbors` value greater than 40 to provide predictions of approximately similar quality.

Adaboost classifier

For this classifier, we decided to test different values of the hyper-parameter `n_estimators` only, which correspond to the number of weak estimators considered by the ensemble method algorithm.

We hence generated the *accuracy score(number of estimators)* curve of FIGURE 3.3.

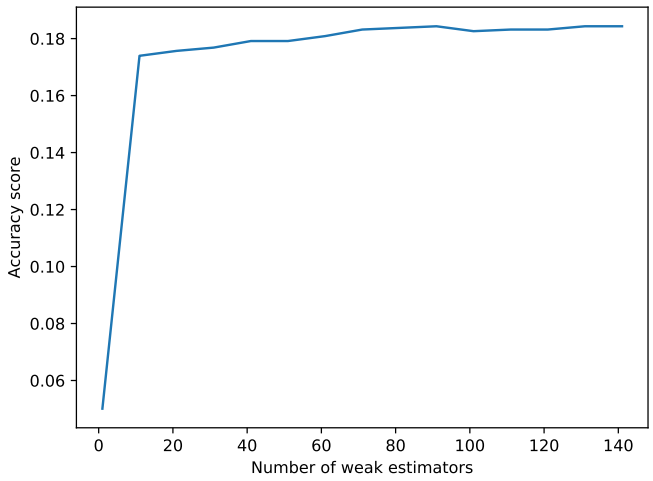


FIGURE 3.3: *Adaboost* classifier : `n_estimators` selection curve

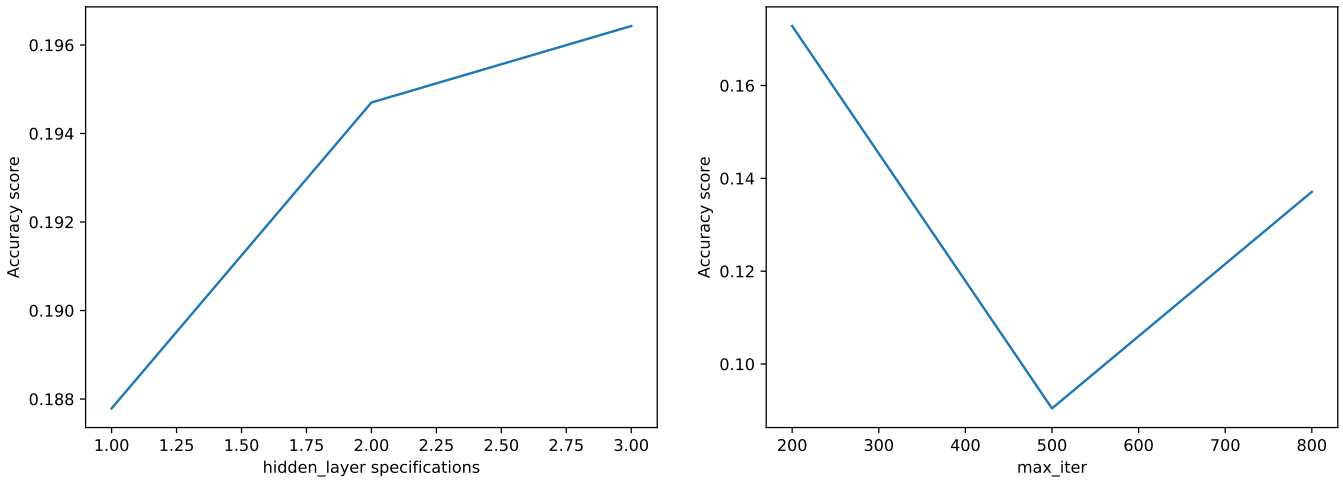
The test set method provided a best `n_estimators` value equal to 91, for a predicted score of 20.39%. However, we can again notice that the accuracy curve reaches kind of a plateau, and `n_estimators` values greater than 20 are expected to provide predictions of approximately similar quality.

MLP classifier

For this classifier, we decided to test different values of the hyper-parameters `hidden_layer_sizes` and `max_iter`, corresponding to both the number of hidden layers and their size, and the maximum number of iterations of the solver respectively.

We hence generated the *accuracy score(hidden layer(s) specifications)* plot of FIGURE 3.4a and the *accuracy score(maximum number of iterations)* plot of FIGURE 3.4b. Please note that FIGURE 3.4a is not continuous: x-axis values 1, 2 and 3 actually correspond to `hidden_layers_sizes` tuples (100,), (100,100,) and (100, 50,) respectively, that is one hidden layer of 100 neurons, two hidden layers of 100 neurons each, and 2 hidden layers of 100 and 50 neurons respectively. Furthermore, since the multi-layer perceptron is quite slow to learn, we only tested 3 hidden layer(s) specifications, alongside with only 3 `max_iter` values.

The test set method provided a best hidden layer(s) specifications setting of two layers of 100 and 50 neurons respectively, giving a predicted score of 19.76%, and a best `max_iter` value of 200 for a predicted score equal to 20.33%. However, since these MLP tuning tests were not very detailed, we did not draw any pertinent conclusion.



(a) hidden layers specification candidates (b) maximum iteration candidates

FIGURE 3.4: MLP classifier : model selection curves

From all these results, we should logically choose *Adaboost* or MLP as learning algorithms for further improvement trials because they seem to work well for the given problem, although the results of the different methods are not drastically different. However, the low predicted scores (as those obtained on the Kaggle platform) caught our attention and we decided to focus on random forests for further investigation, as it is a quite fast and easily tunable algorithm. Indeed, we had the intuition that an underlying problem invalidated our results, and we decided to work with random forests only in order to dig into it.

3.2.2 Improvement attempts and error discovery

Our trouble to achieve good prediction could be due to two reasons. A lack of information given to the model, or a coding error. Actually, we figured out that the inaccuracy was caused by both of these errors. The coding error was in the the `make_pairs_of_players()` function. The output associated to each pair of players was shifted by one unit, and, as a result, when training a model, it was fitting false data. Unfortunately, this has been discovered after the Kaggle competition deadline.

In addition, several features have been added to the model and some of the current ones have been modified to avoid misinterpretation of the algorithm during the training phase. In particular, the feature 2.2.3.4 which defines the zone of a player has been replaced by a series of 5 Boolean features corresponding to each zone. This choice was discussed with Prof. Geurts, as it allows neural network an easier learning when having a 0-1 value for a feature compared to a continuous value. This choice was motivated by our desire to create information that fits to each model. Moreover, the feature 2.2.2.5 which indicates the distance between the pass line and the closest opponent of the sender has been updated as it was discussed in the specific section.

The new features added to the model being numerous, they have been listed here below:

- The distance between the receiver and his two closest team mates (two features, 2.2.2.2)
- The distance between the receiver and his two closest opponents (two features, 2.2.2.3)
- The distance between the sender and the center of the field (2.2.2.6)
- The distance between the receiver and the center of the field (2.2.2.6)
- Gain on the abscissa of the ball (2.2.2.7)
- Ordinate displacement of the ball (2.2.2.8)

- Predicate whether the sender is performing an attack (2.2.3.2)

Once all these features have been added, before the code error discovery, the accuracy predicted following the method explained here above barely changed. This was very confusing. However, we did not gave up, and once the error in the code has been discovered and correctly edited, we obtained pretty good results. The final accuracy curve is depicted on FIGURE 3.5

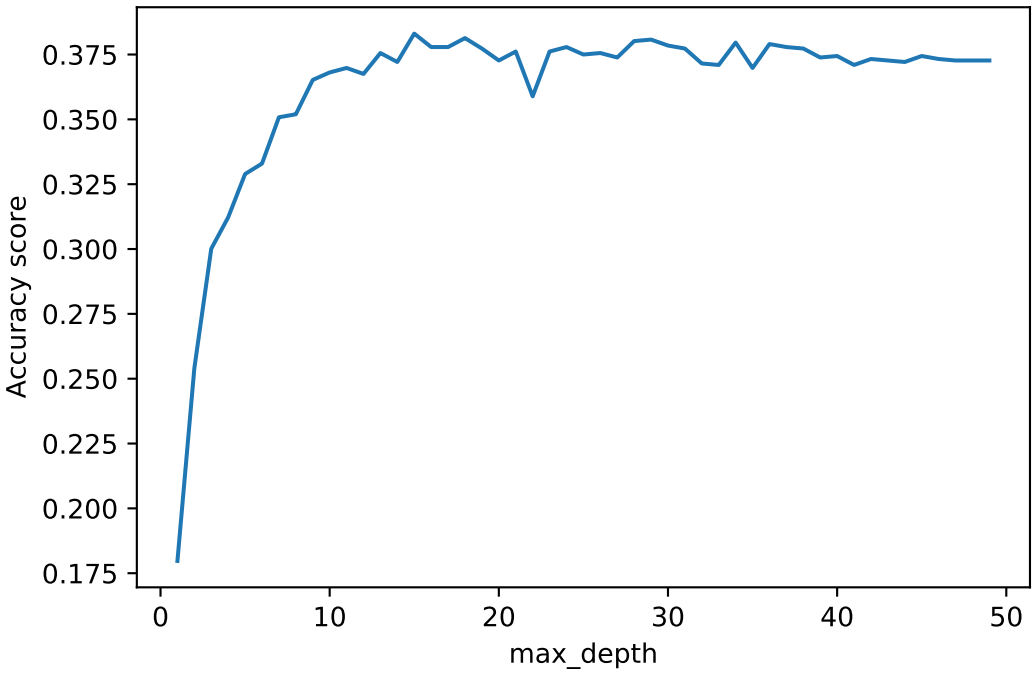


FIGURE 3.5: Accuracy of the random forest model depending on the maximal depth

The best model was found on a depth of 15. The test set method has been achieved in order to predict the performance of the model, the predicted accuracy was 0.3698. Actually, this prediction was very accurate. Indeed, a late submission on the Kaggle platform indicated us that the performance achieved on the whole test set is 0.3705. As the error was found after the competition deadline, we could not select it as a final score. In order to prove what we are saying and to show that our methodology was right, two screenshots have been added on FIGURE 3.6 and FIGURE 3.7.

```
Performance estimate: 0.36981566820276496
X_LS_VS_TS is of shape (191004, 25)

Training on LS+VS+TS...

Predicting...
(3000, 47)
Deriving predictions from probabilities

Submission file "Data/random_forest_test_set_methodall_features_13-12-2020_18h41.txt"
successfully written
```

FIGURE 3.6: Estimation of the performance of the final random forest model

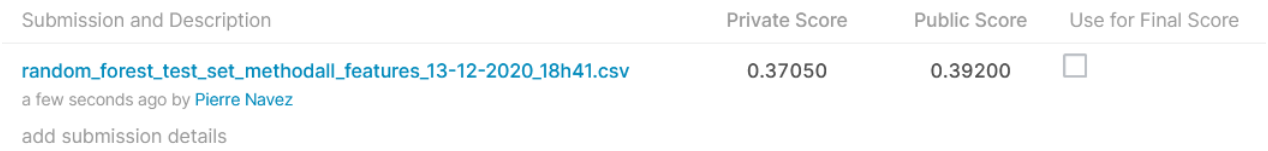


FIGURE 3.7: Real performance based on the submission on the Kaggle plateform

3.3 Task separation during the project realization

During the project realization, we did not assign a particular task to everyone but we were going forward together. However, Guilian Deflandre and Pierre Navez were already forming a team during the two first projects and Antoine Debor joined this team as its previous collaboration with another teammate was unsuccessful due to the bad separation of the work.

Nevertheless, we can briefly summarize in practice how the work as been achieved. Antoine coded some of the tools as the distance matrix, used is all features derivation function. During this time, Guilian and Pierre investigated on the net about which feature could provide interest. Everyone has coded some of the features. Antoine implemented the test set method and when it has been achieved, everyone of us led some test on different algorithms. The report was evenly separated.

4 Conclusion

As a final conclusion, one can say that the challenge was not easy at all. In fact, it was very frustrating to deal with the bad results obtained without knowing where they came from. This took us a lot of time to debug the code and once the error has been found, we faced a lack of time to improve our results at their finest. However, one can enumerate some improvements that we think, would have provided us better results. Besides testing several models such as adaboost, a neural network or a gradient boosting algorithm, some new features of interest could have been added. For instance, features that capture the game state like, is the potential receiver team defending? Also, features about the pass feasibility such as the minimum angle between a possible pass line and an opponent. Additionally, we did not take into account that some players of the data set were actually out of the field and we suppose that they were out of the game, possibly due to a red card or medical reason. These details would have provided more pertinent information to the learning algorithm. Another refinement that could be performed would be to test different combinations of the whole set of features to extract essential ones and drop redundant ones.

5 Acknowledgments

We would like to sincerely thank Prof. Geurts and Antonio Sutera for their availability and their time they gave us during the project to answer our questions especially because it was one or two days before the final deadline.

References

[1] Y. Dauxais and C. Gautrais, *Predicting pass receiver in football using distance based features*, in International Workshop on Machine Learning and Data Mining for Sports Analytics, Springer, 2018, pp. 145–151.

[2] P. Fournier-Viger, T. Liu, and J. C.-W. Lin, *Football pass prediction using player locations*, in International Workshop on Machine Learning and Data Mining for Sports Analytics, Springer, 2018, pp. 152–158.

[3] O. Hubáček, G. Šourek, and F. Železnỳ, *Deep learning from spatial relations for soccer pass prediction*, in International Workshop on Machine Learning and Data Mining for Sports Analytics, Springer, 2018, pp. 159–166.

[4] M. Kesson, *Vectors - shortest distance from a point to a line*. Online, `http://www.fundza.com/vectors/point2line/index.html`, 2002.

[5] The Machine Learning and Data Mining for Sports Analytics workshop organizers, *5th workshop on machine learning and data mining for sports analytics*. Online, `https://dtai.cs.kuleuven.be/events/MLSA18/`, December 2020.