Université de Liège

Master in engineering

December 13, 2020

ELEN 0062
Introduction to Machine Learning

Project 3

Next pass prediction in a football match

Maxime Amodei - **s171830**
Justin Moreels - **s145505**
Cédric Siboyabasore - **s175202**

1 Introduction

Sports analytics is a field that has grown popularity in the machine learning community in recent years. As a consequence, yearly conferences have been organized by the European Conference on Machine Learning (ECML) and the European Symposium on Principles of Knowledge Discovery and Data Mining (PKDD) [1].
In this project inspired by the Football Pass Prediction Challenge of the 5th Workshop on Machine Learning and Data Mining for Sports Analytics and organized as a Kaggle competition [2], our goal is to use machine learning techniques to predict the next receiver of a pass during a football game. This challenge is the occasion for us of applying the machine learning techniques we learned in class.

2 Exploring the data

To solve the problem we are provided with a dataset of 8682 samples corresponding to passes and for each of these samples information is provided about the time since the start of the halftime the ball was passed, the 22 players positions on the field at the time of the pass and the IDs of the sender and receiver players of the pass.
We were also given a `toy_example.py` file. In this file the supervised learning problem is defined such that we have to solve a binary classification problem where we have 2 players (a sender and a candidate receiver out of the 22 players) and the binary classification problem consists in predicting whether or not the candidate receiver is indeed the pass receiver. The new dataset therefore contains $8682 * 22 = 191004$ samples.
Solving the problem this way seems much less complex than solving the problem as a multi-class classification problem with 22 outputs corresponding to the 22 players so in this project we will mostly keep the problem defined as a binary classification problem.
In the file a Decision tree classifier was built upon two features computed from the dataset: the (Euclidean) distance between the pass sender and the candidate receiver and whether they're part of the same team or not. After predicting the class probabilities of the input data of a test set of size 3000 with this Decision tree model we obtain a poor public score of **0.14599** on the Kaggle platform.
Building a decision tree based on only 2 features may not be enough to minimize the misclassification error reaching the leaves of the tree. Some pre-processing of the data can be performed and other binary classification models can be tried after adding other features.

3 Pre-processing of the data

The football pitch is 105 meters long ($x \in [-5250cm, +5250cm]$) and 68 meters wide ($y \in [-3400cm, +3400cm]$) but some x and y player position coordinates were found in the data, exceeding the y coordinates interval meaning that some players were outside of the football field.
We thus arbitrarily chose to strip the original dataset of the samples/snapshots where a player was exceeding the x bounds by 250 cm (2.5 m) or the y interval by 150 cm (1.5 m). The number of eliminated samples is 215 and it's reasonable compared to the total number.

4 Features approach

We decided to dig deeper in the features approach by creating new ones for a better performance for our models. The goal was to have more features describing the interest of a pass for the sender but also its difficulty. The more complicated a pass is to make, the greater the chances that the receiver is an opponent.
This part of a machine learning problem requires domain knowledge which we had since all members of our team have played football and know the rules of this worldwide famous sport.
The final features are detailed hereafter:

1. Distance between the sender and the potential receiver: obviously, the closer a player is, the more likely he is to get the ball. However, long distance passes also happen. This feature will help to characterize them. This feature was already provided in `toy_example.py`.

2. Mean position of the sender's team: the position of the team can influence the difficulty and the interest of a pass.

3. Mean position of the opponent's team: Same reasoning for the opponent's team.

4. Side of the sender's team

5. Forward distance: corresponds to the difference between the x-coordinate of the player and the x-coordinate of the sender. This distance characterizes the interest and the difficulty of a pass, either an advancing one or backward one.

6. Lateral distance: corresponds to the difference between the y-coordinate of the player and the y-coordinate of the sender. This distance can again influence the interest and difficulty of this pass.

7. Intercept distance: the shortest distance of an opponent of the sender to a line segment between the sender and the potential receiver. Only the opponents located between the two players involved in the pass are taken into account. Indeed, opponents located behind these players can not intercept the ball.

8. Distance between the sender and an opponent: a close opponent increases the difficulty of a pass.

9. Distance between the potential receiver and an opponent of the sender: again, a close opponent increases the difficulty and the potential interception.

10. X-zone of the sender: the pitch was divided in five zones along the x-axis. The closer the sender is to his goal, the smaller is the number of his X-zone. In a way, it represents the advance of the pass such as the forward distance. Nevertheless, this feature is much less precise but take in account the initial position of the sender, which affect the objectives of the sender and his willingness to take risks.

11. Y-zone of the sender: the pitch was divided in three zones along the y-axis. The choices of the sender differ in function of his y-location. For example, it is easier to make a pass to a lateral zone when the sender is already in the same lateral zone.

12. X-zone of the potential receiver: feature related to the tenth feature.

13. Y-zone of the potential receiver: feature related to the eleventh feature.

14. Same team: boolean representing if the sender and the potential receiver are in the same team. It will obviously influence the sender decision. This feature also makes it possible to categorize all the passes to the opponent's team. This feature was already provided in `toy_example.py`.

5 Models

5.1 Model testing with default parameters

Now that the features are defined, a classification model has to be chosen. To that end, several estimators from the *sklearn* package have been tested with their default parameters.
To measure and compare the performances of these classification models we used the Area Under the Receiver Operating Characteristic curve (ROC AUC) score because we mainly care about ranking predictions [3]. We evaluated it for each model through cross-validations by using function `cross_val_score` with its `scoring` parameter set to 'roc_au' [4]. Table 1 allowed us to eliminate some of the models:
We first attempted to improve the Decision tree of `toy_example.py` with the pre-processed data and the new features but as anticipated the model remains too simple as the AUC score is low. Moreover, the K neighbors classifier, the Linear SVC, the SGD classifier and the Gaussian NB are less performing than the others. The SVC requires a lot of computing time so it was also put aside (more details in section 5.2.1). We made the choice to look further into the Random forest, Adaboost, Gradient boosting and MLP classifiers. We also decided to first try to optimize the value of its `n_neighbors` parameter of the such as we did in Project 1.

Model	AUC score	Variance
Decision tree	0.59	0.01
Random forest	0.84	0.01
K neighbors classifier	0.71	0.01
Linear SVC	0.64	0.22
SVC(`kernel`='rbf')	0.85	0.01
SGD classifier	0.78	0.02
Gaussian NB	0.81	0.02
Gradient boosting classifier	0.85	0.01
Adaboost classifier	0.84	0.01
MLP classifier	0.84	0.01

Table 1: AUC scores for different models with default parameters

5.2 K-nearest neighbors

In Table 1 we used a K-nearest neighbors (KNN) classifier with the `n_neigbors` parameter set to its default value `n_neigbors`=5.
To obtain the optimal value of `n_neigbors` we chose to do a five-fold cross validation strategy, just like in project 1: we built several nearest neighbors models with values of `n_neighbor` ranging from

1 to 100 and estimated the mean AUC score of each K-nearest neighbors model on the features by splitting the features into a learning set (4 subsets of the features) and a test set (remaining subset), fitting the model and computing the score 5 consecutive times with different splits each time using the `cross_val_score` function from the `sklearn.model_selection` module. For each of the models we stored the mean AUC score over the 5 splits in an array in order to find the `n_neighbor` parameter that corresponds to the maximum mean score.

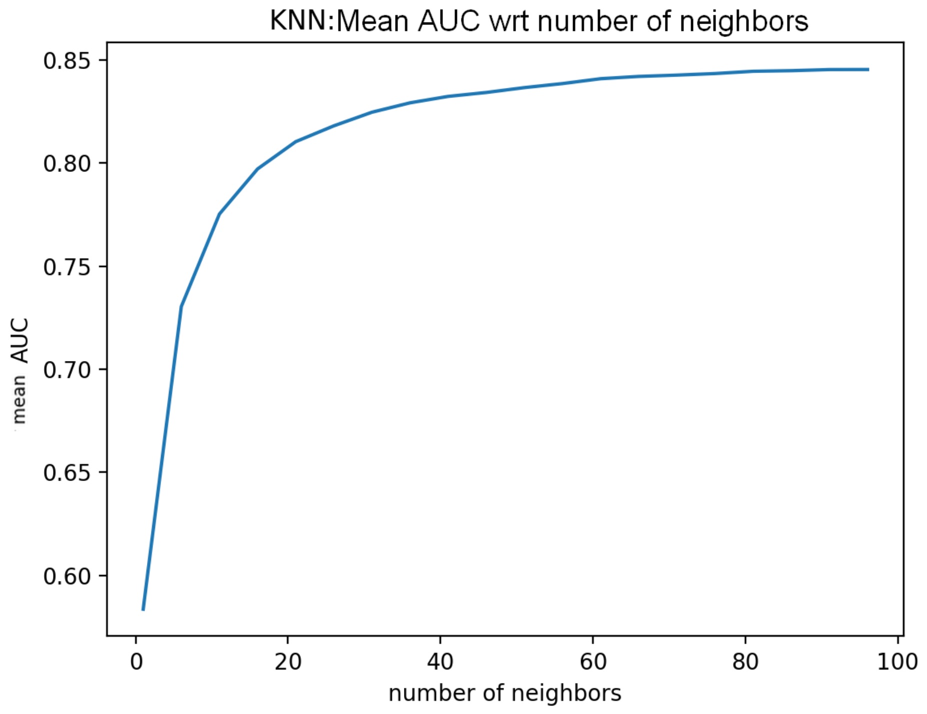

Figure 1: Evolution of the mean AUC score over the five folds for KNN

The maximal mean AUC score 0.846 (+/- 0.007) is obtained when `n_neighbor = 100`.
We then built a K-neighbors classifier with `n_neighbor = 100`, fitted the model on all of our features and derived the probability estimates for the input data of the test set.
With this method we obtain a public score of **0.36300** on the Kaggle platform.
For the next classification models, we will compute the mean AUC score by performing 4-fold cross validations in order to save up computation time. 4 is a good trade-off between a sufficient number of splits of samples and a not too long computation time.
It it worth mentioning that we will also evaluate our next binary classification models on their AUC score and on their public Kaggle score.

5.2.1 Support vector machine

Although we reached a higher public Kaggle score by using the K-nearest neighbors classifier with an optimal `n_neighbor` parameter, as the number of features has increased from 2 to 14 we might have to think about other models. With a high dimensional features space the K-nearest neighbors model might suffer from the curse of dimensionality : looking at the neighbors of a new object in the

test set might be troublesome because the neighbors are so far away that they might not be provide the right prediction. A supervised learning model that behaves well with high dimensional spaces is the Support Vector Machine model for binary classification in this project.
In Table 1 we used a Support Vector Classifier (SVC) with a default `kernel` parameter set to 'rbf' (radial basis function) and a regularization parameter `C=1`. This rbf Support Vector Classifier yielded a higher AUC score than the linear kernel Support Vector Classifier (`C=1`) which provided an AUC score of 0.64 (LinearSVC is similar to SVC with parameter `kernel=`'linear') [5].
We also wanted to compare the AUC scores of the rbf SVC with a polynomial kernel (`kernel`=poly) SVC for a common regularization parameter `C`=1 but were unable to do it as support vector machines have a complexity order of $\mathscr{O}(n_{features} * n_{samples}^2)$ at training time [6] [7]. As we have 14 features and 8467 samples, the execution never ended on either of our computers.
We would also have wished to vary the parameters `gamma` and `C` of the SVC for the rbf kernel. Indeed, `C` interacts strongly with `gamma` [8] [9] and it would have been interesting to perform a `GridSearchCV` ("algorithm of the *sklearn* package that performs exhaustive search over specified parameter values for an estimator" [10]) with cross-validation to find the best `gamma` and `C` parameters combinations that would lead to the highest AUC score.
However, we couldn't tune the parameters and see the results because of the time cost.

5.3 Adaboost classifier

One way to improve the performance of the Decision tree classifier was to boost decision trees with the Adaboost classifier. Indeed, boosting with Adaboost using a shallow decision tree as base estimator was referred to as "the best off-the-shelf classifier in the world" by distinguished statistician Leo Breiman [11].
Using the Adaboost classifier with its default parameters we fitted `n_estimators`= 50 consecutive Decision trees of depth 1 and obtained an AUC score of 0.84. This is an improvement with respect to the score we obtained for the sole Decision tree model.
To get an even better score, we tried to find a number of estimators that would lead us to an optimal AUC score by using four-fold cross validation on Adaboost classifiers we built still from a decision tree of depth 1 but with a number of estimators ranging from default 100 to 800 by steps of size 100 (to decrease computation time).

As we can see on Figure 2 representing the evolution of the AUC score with respect to the number of estimators, the AUC score didn't vary much from the value 0.84 that was reached when the number of weak estimators was at its default value `n_estimator`=50. It reaches its peak when `n_estimators`=400 on the graph for a slightly increased AUC of 0.8433, still with a variance of 0.01.
In other words, the default number of 50 decision trees of depth 1 was already enough to bring us close to the optimal AUC score for the AdaBoost classifier.
With `n_estimators`=400 we obtained a public Kaggle score of **0.35200**.

5.4 Gradient boosting classifier

The Gradient boosting classifier obtained the best AUC score in the initial comparison (see Table 1. Therefore, we decided to tune its parameters using the `GridSearchCV`.[12] This algorithm also need some parameters. AUC score is still used to evaluate the performance and we chose 5 folds for the

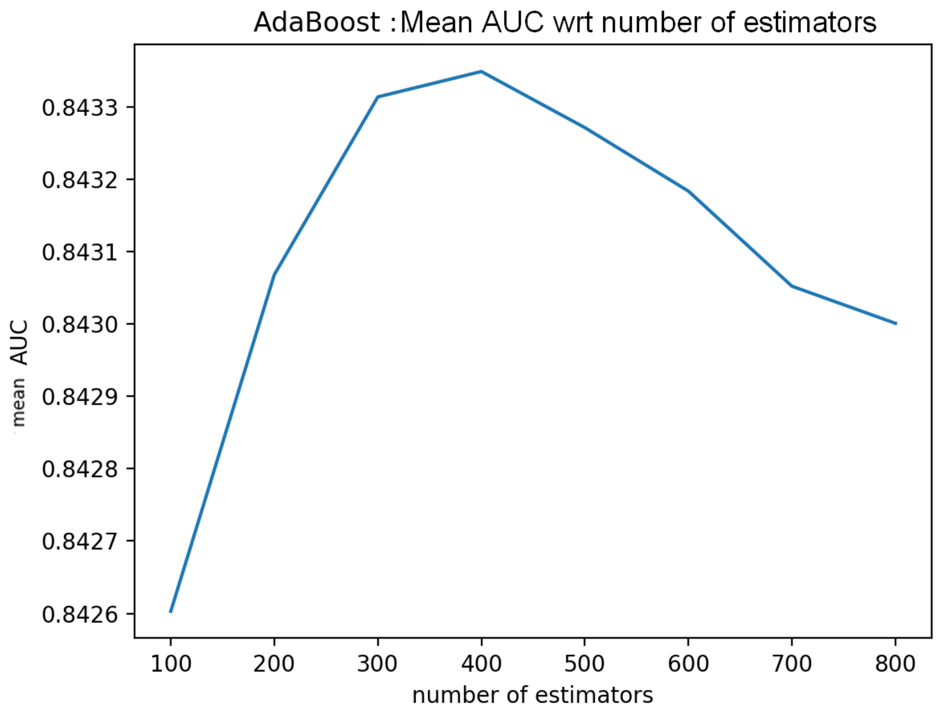

Figure 2: Evolution of the mean AUC for the AdaBoost classifier

cross validation. Before starting, we can look at the importance of each feature with default settings. Figure 3 shows that only few features are used to derive the model so it can be improved.

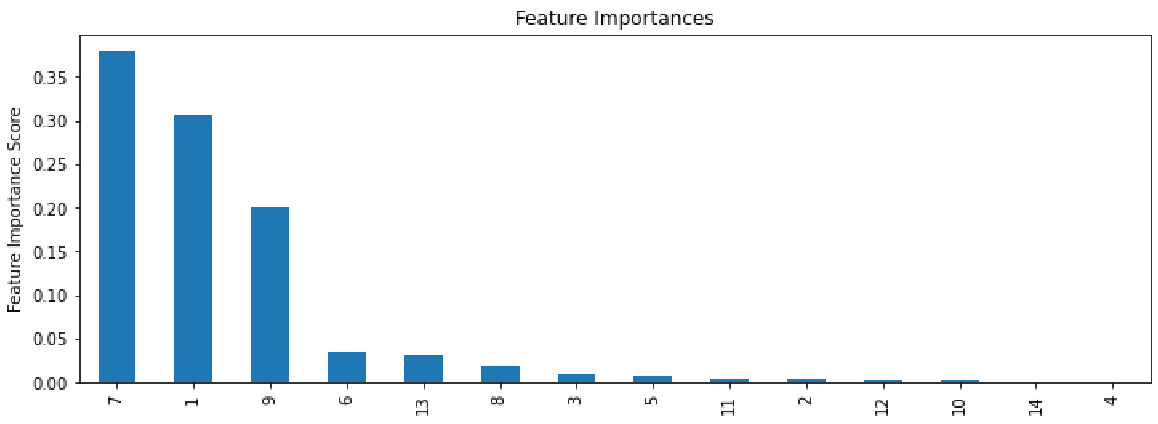

Figure 3: Initial importance of features with the Gradient boosting classifier.

1) Number of trees and learning rate

The optimal number of trees for a learning rate of 0.1 was the upper bound of the first tested values (`n_estimators` = 140). To decrease the computation time during the testing, a learning rate of 0.2 was set. The new optimal number of trees is then 80. After tuning all parameters, the learning rate will be reduced.

2) Depth of a tree and minimum number of samples required to split

These two parameters, `max_depth` and `min_samples_split` are tested together. Too great depth or too small minimum can lead to overfitting. On the other hand, too high values for `min_samples_split` can lead to underfitting. As a result, we get `max_depth` equal to 7 and `min_samples_split` equal to 1400, which is reasonable.

3) Minimum number of samples in a leaf

This parameter is tested with several values of `min_samples_split` (around the previously found optimal number). Finally, `min_samples_leaf` can be set to 60.

The importance of each feature can be displayed again to show the evolution. Indeed, Figure 4 highlight a better distribution of importance among the features and more variables are taken in account.

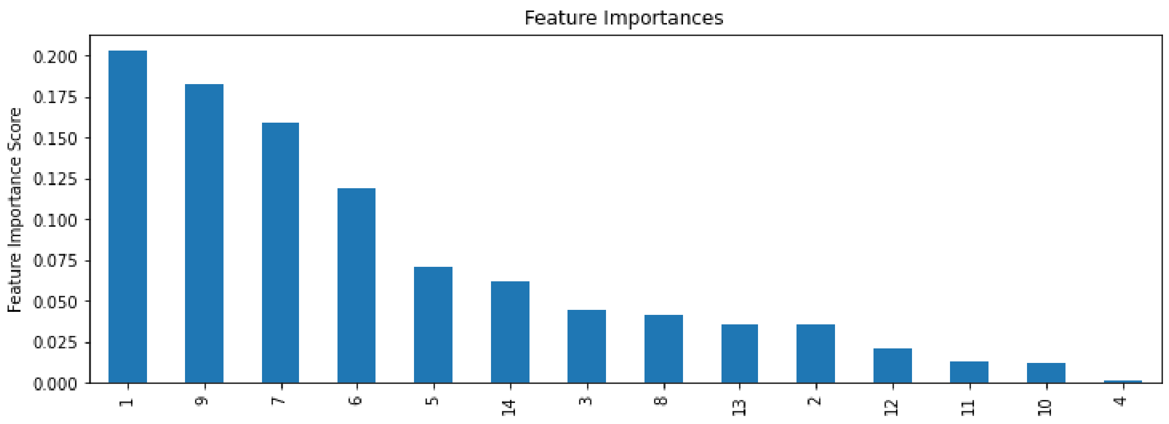

Figure 4: Importance of features at step 3 with the Gradient boosting classifier.

4) Maximum number of features considered to make a split

The value of `max_features` was initially set to the square root of the number of features. This is a common way to proceed. By using `GridSearchCV`, the optimal value can be set to 3. It roughly correspond to the initial value because there are 14 features.

5) Subsample parameter and finalisation

This parameter represent the fraction of samples to be used for fitting the individual base learners. A value below 1.0 leads to a reduction of variance and an increase in bias. A commonly used value is 0.8. We find our optimal value at 0.75.

To increase performances of the model, the learning rate can be decreased while the number of estimators can be increased proportionally. The initial learning rate of 0.2 is reduce by half while the initial number of estimators of 90 is multiplied by 2. These operations are repeated three times, until the AUC score no longer improves significantly. At the end, we get a learning rate of 0.025 and 720 estimators.

The public score on the Kaggle platform of this model is **0.385**. This score being our second best and the choice of parameters having been made paying attention to overfitting, **this model is the first submission retained**.

5.5 Random forest classifier

Similarly to the tuning on the parameters of the Gradient boosting classifier, an optimization has been performed on the Random Forest classifier with `GridSearchCV`. The type of parameters is also similar because both classifier use trees. The final optimal values are listed below :

- `n_estimators` = 100
- `min_samples_leaf` = 50
- `max_leaf_nodes` = 350
- `min_samples_split` = 200
- `max_depth` = 13

In contrast with the Gradient boosting classifier, the Random forest has less potential because the importance of each feature is already well distributed with the default parameters (see Figure 5). Confirming our intuition, the public score of this model on the Kaggle platform is **0.365**, which is below the score of the Gradient boosting model.

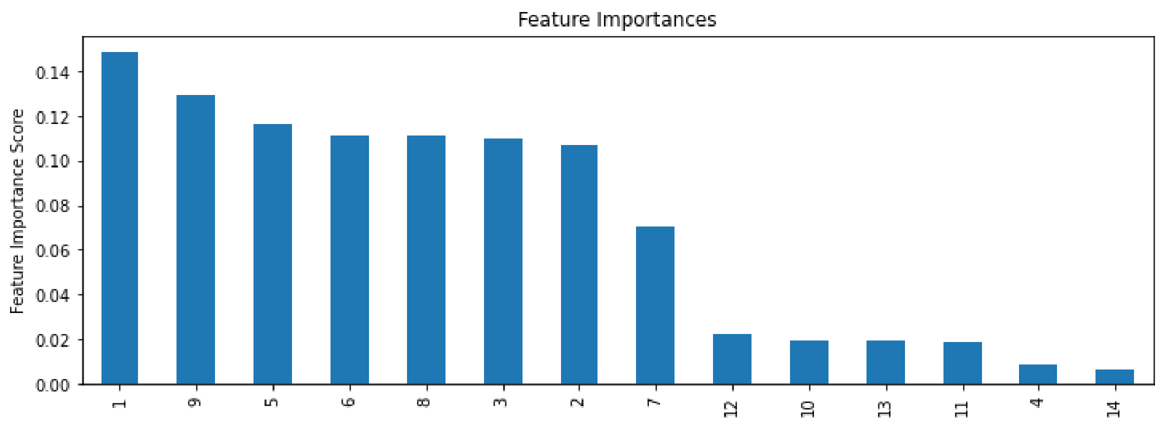

Figure 5: Initial importance of features with the Random forest classifier.

5.6 MLP classifier

The Multi-layer Perceptron classifier is a different kind of model because it uses neural networks and not on ensemble like previous ones. Again using the `GridSearchCV` function, we looked after the best parameters.

First, the number of neurons in each layers has to be defined. Several values for the number of layer and the number of neurons per layer has been tested. Finally, we set the number of layer to three with a decreasing number of neurons : `hidden_layer_sizes` = (150,100,50). Then, several parameters are tested but the default one is still the better choice : `activation`, `solver`, `alpha`, `learning_rate` and `max_iter`.

The final model obtained a public score of **0.386** on the Kaggle platform. Given that this is our best score and that there was no choice that could lead to overfitting, **this is the second submission retained**.

6 Deep Convolutional Neural Network

We suppose that this task could be solved easily by a football fan watching the game and paying attention to the disposition of the players in the football field: the spacial structure is a crucial set of features of the problem and yet it is very hard to fully describe. A fan watching the game also has access to the previous moments before the pass and those moments capture important features, such as the velocity of the players. This reasoning motivates the use of a computer vision solution : we constructed an artificial image representing the data with all the player position and highlighting the sender position. The first team was colored in red, the second team in blue and the sender was added with some green such that player super-positions are easily noticeable. The computer vision task was tackled with image segmentation: the model was designed to highlight the possible positions for the receiver of the pass in the image. We found some inspiration in the Unet model [13] which made a breakthrough in image segmentation, and there is a variant of Unet in [14] that was shown to be capable of locating some small object on an image by higlighting them so we built a similar architecture in the hope that the model would be able to highlight the receiver.

As the dataset has a few sample where the player are outside of the playing field (by up to 5m) we included safety margin in the image (4.1m horizontally and 5.6m vertically) to represent all positions in the training set. The image was reshaped to have a size of 64 by 64 pixels to save some memory usage. Similarly the label for each data was created in a 64 by 64 image using the same safety margins with only one pixel highlighted representing the position of the receiver. These constructions allow us to easily represent the spacial features but it is now up to the model to identify them and make some predictions based on the images which needs a complex model. Moreover this model requires some post-processing to extract the probabilities that each player receives the ball: to estimate this distribution we look a the neighboring pixels corresponding to the position of each player and compute a weighted sum on the neighboring pixels which can be interpreted as a (scaled) probability measure. All those probabilities can then be normalized to produce the final output.

Deep neural networks are hard to train: they typically have a high variance and are prone to over-fitting due to their high complexity. We used techniques such as dropout, weight decay and early stopping to avoid over-fitting and obtain a good idea of the performances of the model. For CNN the use of regular dropout layer is not encouraged because neighboring values may already share the same information and the Independence is often preferred to be between the different channels, for this reason we used the dropout2d technique which removes a channel entirely instead of just some elements.

We first trained the model with the Adadelta optimizer which adapts the learning rate automatically then we trained the model with a lower learning rate and slightly higher weights decay to improve the performance and avoid over-fitting. The loss and accuracy of the model over the training and testing data sets for one model is shown in Figure 6: we can see the the performance improve at first but after epoch 60 the test loss starts to increase which is a clear sign of over-fitting. The accuracy over the test set hover between 25% and 27% in the second part which is a sign that the model isn't picking up the right features since some other models performed much better. These values gave us a good idea of the potential performance of the model, so we trained a new one and stopped the training early (at around 40 epochs) and the score of this model on Kaggle was **0.271** so the estimate on the test and validation set were accurate, however the model seemed to be under fitting and all attempts to train it further lead to over-fitting.

Finding a good structure for the model was challenging: we got some inspiration from the Unet to use a fully convolutionnal layers divided in a contraction and an expansion part with the later re-using some feature extracted by the former. Our structure was adapted to work with smaller image and less

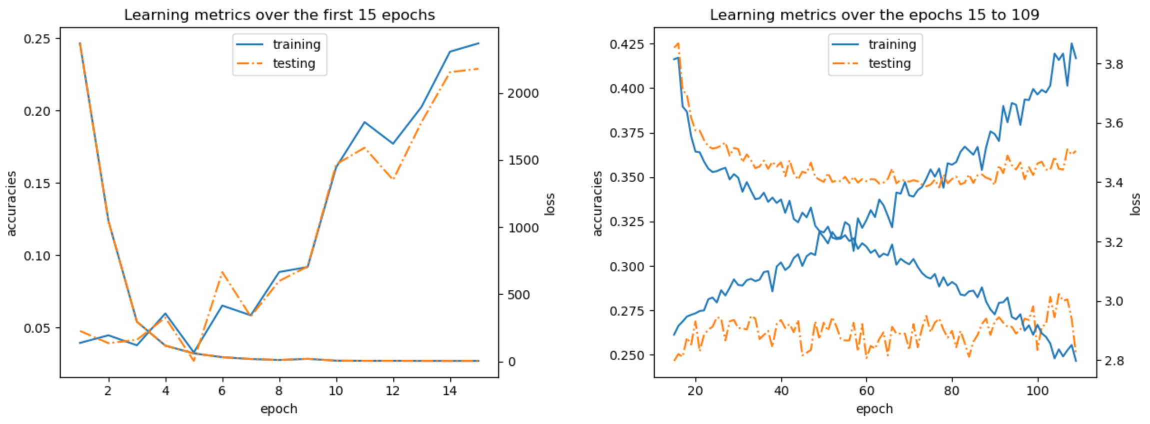

Figure 6: Training the convolutionnal neural network in two parts.

feature for the model to train faster, we think that improving the depth of the model or the number of feature could potentially improved the solution but weren't able to train a significantly better model by tweaking those values.

7 Conclusion

The goal of the project was to predict passes during a football match. Two approaches were studied.

The first one was based on several features describing a pass between the sender and a potential receiver. The probability of receiving the pass for each player was computed and the candidate player associated to the highest one is considered to be the receiver. To have a first idea of the learning algorithm that would be appropriate, the most common classification models were tested with the full set of features and their default parameters. Five of them were retained and their parameters were tuned to optimize their performance. Finally, the best results were obtained with the Gradient boosting classifier and the MLP classifier. The first one using trees had a public score of **0.385** and a *private* Kaggle score of ***0.371***. Similarly, the second model using neural networks obtained a public score of **0.386** and a *private* score of ***0.374***.

A second approach based on deep convolutional neural network has been tried: we designed a model inspired from a state of the art architecture and trained it carefully to avoid over-fitting. However this model didn't perform very well and failed to learn all the relevant features, but as it is a black-box model the precise causes of the lower performances are hard to spot. The model obtained a modest score of 0.271.

As a conclusion, this project allowed us to use and become more familiar with several learning algorithms. The final scores obtained are not really high so had we had more time we think we would have been able to reach higher value. Some new features that better take into account characteristics of failed passes would probably be a good start.

Contribution

- Maxime Amodei : Deep learning techniques : a fully convolutional neural network and a simpler one that was abandoned because we couldn't train it.
- Justin Moreels : Feature engineering + Gradient boosting classifier + Random forest classifier + MLP classifier.
- Cédric Siboyabasore : Feature engineering + K-nearest neighbors classifier + Support vector machine + Adaboost classifier

References

[1] *5th Workshop on Machine Learning and Data Mining for Sports Analytics*. URL: `https://dtai.cs.kuleuven.be/events/MLSA18/`.

[2] *Next pass prediction in a football match*. URL: `https://www.kaggle.com/c/iml2020/`.

[3] *F1 Score vs ROC AUC vs Accuracy vs PR AUC: Which Evaluation Metric Should You Choose?* URL: `https://neptune.ai/blog/f1-score-accuracy-roc-auc-pr-auc`.

[4] *sklearn.model_selection.cross_val_score*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.cross_val_score.html`.

[5] *sklearn.svm.LinearSVC*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html`.

[6] *StackExchange: SVM using scikit learn runs endlessly and never completes execution*. URL: `https://datascience.stackexchange.com/questions/989/svm-using-scikit-learn-runs-endlessly-and-never-completes-execution`.

[7] *stackoverflow: Why is scikit-learn SVM.SVC() extremely slow?* URL: `https://stackoverflow.com/questions/40077432/why-is-scikit-learn-svm-svc-extremely-slow`.

[8] *RBF SVM parameters*. URL: `https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html`.

[9] Kevin P. Murphy. "Machine Learning: A Probabilistic Perspective". In: 2013, p. 504.

[10] *sklearn.model_selection.GridSearchCV*. URL: `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV`.

[11] Kevin P. Murphy. "Machine Learning: A Probabilistic Perspective". In: 2013, p. 554.

[12] *Parameter Tuning in Gradient Boosting*. URL: `https://www.analyticsvidhya.com/blog/2016/02/complete-guide-parameter-tuning-gradient-boosting-gbm-python/`.

[13] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: `1505.04597 [cs.CV]`.

[14] Javier Ribera et al. *Locating Objects Without Bounding Boxes*. 2019. arXiv: `1806.07564 [cs.CV]`.