DataScientest • com

Rapport projet « paris sportif »

Bet-py

Promotion Data Scientist décembre 2020

Groupe Projet : Matthieu Morand & Romain Miclo

Tuteurs : Maxime, puis Théophile à partir de juillet

Contents

1. Introduction 2
Objectif du projet 2
Données de départ 2
2. Exploration et visualisation des données 3
Présentation des données 3
Visualisation des données 5
3. Nettoyage, mise en forme et choix des features 9
Première itération complète 9
Seconde itération 10
Définition jeux train / test et standardisation des données 12
4. Modèle : création, application et évaluation 13
Une première itération complète 13
Seconde itération avec nouvelles features 14
Conclusion du projet 19
Rappel des fichiers à consulter 19

1. Introduction

Objectif du projet

L'objectif initial de ce projet est de tenter de battre les algorithmes des bookmakers sur l'estimation de la probabilité d'une **équipe gagnant un match**.

Ce projet permet de traiter l'ensemble des étapes d'un projet de Data Science :

- Exploration et visualisation des données
- Nettoyage et mise en forme des données
- Définition et choix des features
- Création, application et évaluation de modèles

Données de départ

Pour ce projet, deux jeux de données étaient proposés :

- Un premier sur le football : https://www.kaggle.com/ayotomiwasalau/club-football-event-data?select=match.csv

- Un second sur le tennis avec les matchs ATP : https://www.kaggle.com/edouardthomas/atp-matches-dataset

Après avoir regardé le jeu de données, nous avons **choisi de partir sur le football** en estimant qu'il y avait un plus grand nombre de données et donc normalement la possibilité d'aller « plus loin » dans ce sujet de « battre les bookmakers ».

Remarque : Il est à noter que nous avions démarré le projet à 3, mais l'un des membres a malheureusement dû arrêter la formation et nous avons donc réalisé ce sujet en binôme.

Les différents notebooks et fichiers réalisés pendant le projet sont cités au fur et à mesure de ce rapport. A la fin de la conclusion, vous avez également le récapitulatif des fichiers et des emplacements de ces derniers, si vous souhaitez les consulter dans notre espace Git.

2. Exploration et visualisation des données

La première phase du projet a eu pour but de décrire les étapes majeures d'exploration et visualisation des données.

L'archive de départ contenait les 5 fichiers suivants avec les différentes colonnes :

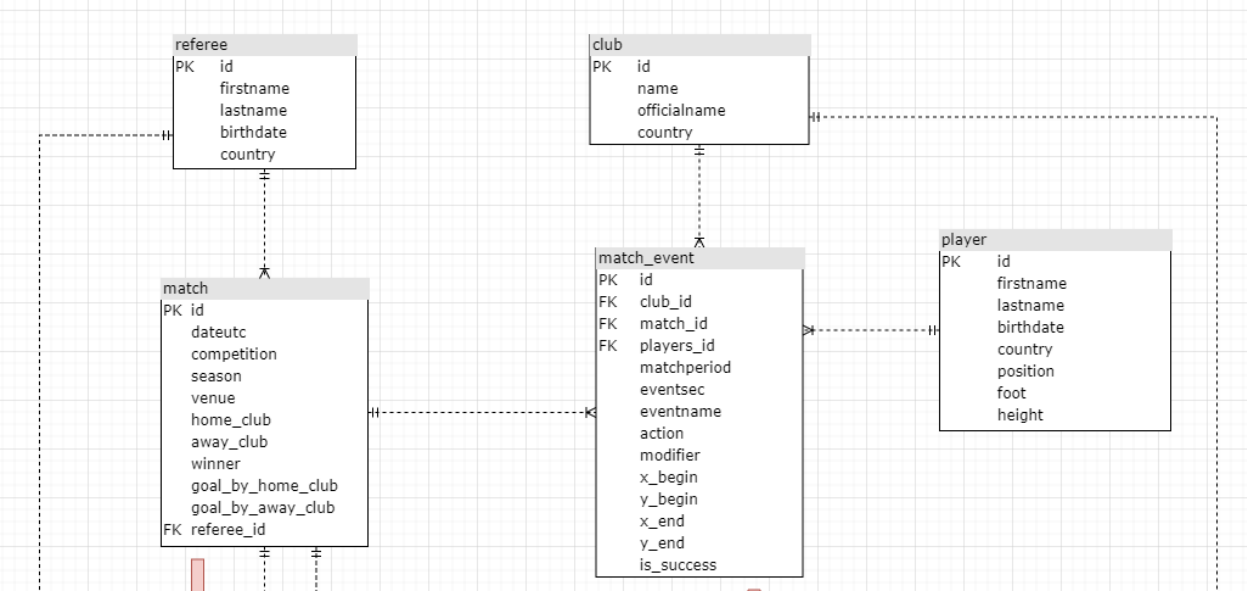


Figure 1: Vue DrawIO des 5 tables de départ fournies

Nous avons rapidement statué qu'à priori « seules » les tables « match » et « match_event » allaient pouvoir nous aider.

La table « match » contient les informations principales résumant le match et surtout **notre label qui est le club gagnant**. La table « match_event » représente quant à elle le cœur des informations disponibles avec des millions de lignes contenant le détail des matchs et permettant donc de créer potentiellement un grand nombre de features.

Présentation des données

Les données fournies présentent les résultats de la saison 2017-2018 des 5 championnats majeurs de football en Europe. Seuls les résultats avec des victoires apparaissent. C'est un point majeur car **les matchs nuls n'y sont donc pas**, c'est pour cela que la base ne contient pas tous les résultats de la saison.

Voici les championnats représentés avec le nombre de matchs associés (on peut par exemple en déduire qu'en plus d'avoir un nombre de clubs moins important en Bundesliga, il y a eu beaucoup de matchs nuls dans le championnat allemand) :

Figure 2: Vue Power BI du nombre de matchs dans la base sur la saison dans les 5 championnats

La table « match » contient les informations de 1 339 matchs et la table « match_event » contient les informations d'environ 2,845M event sur ces matchs.

Au-delà des 5 pays où se jouent les matchs, les joueurs proviennent de nombreux pays différents dont une très grand majorité en Europe puis un grand nombre en Amérique du Sud et en Afrique :

Figure 3: Vue Power BI - nationalité des joueurs ayant joué au moins un match

Cette information n'a pas été retenue pour le modèle mais pourrait être testée (voir si une équipe avec un grand nombre de nationalités ou certaines nationalités est plus performante statistiquement qu'une autre par exemple).

Visualisation des données

Afin de prédire une victoire ou une défaite, une information essentielle est le nombre de buts marqués/encaissés, voici la répartition des buts marqués à domicile et à l'extérieur sur les jeux de donnés (cf. Figure 4).

On peut voir que le nombre de buts marqués à l'extérieur est plus faible sur chaque modalité qu'à l'extérieur (par exemple 19% au lieu de 25% des matchs avec 2 buts). Le fait de recevoir ou de se déplacer est donc déjà une feature importante à prendre en compte dans le modèle.

Afin de tester les futurs modèles, il est important de repérer les clubs ayant le plus de victoires dans leurs championnats respectifs afin de vérifier si l'algorithme se comporte bien pour ces clubs majeurs, tels que la Juventus en Italie, le PSG en France ou Manchester City en Angleterre (cf. Figure 5, 3 premiers clubs en abscisse).

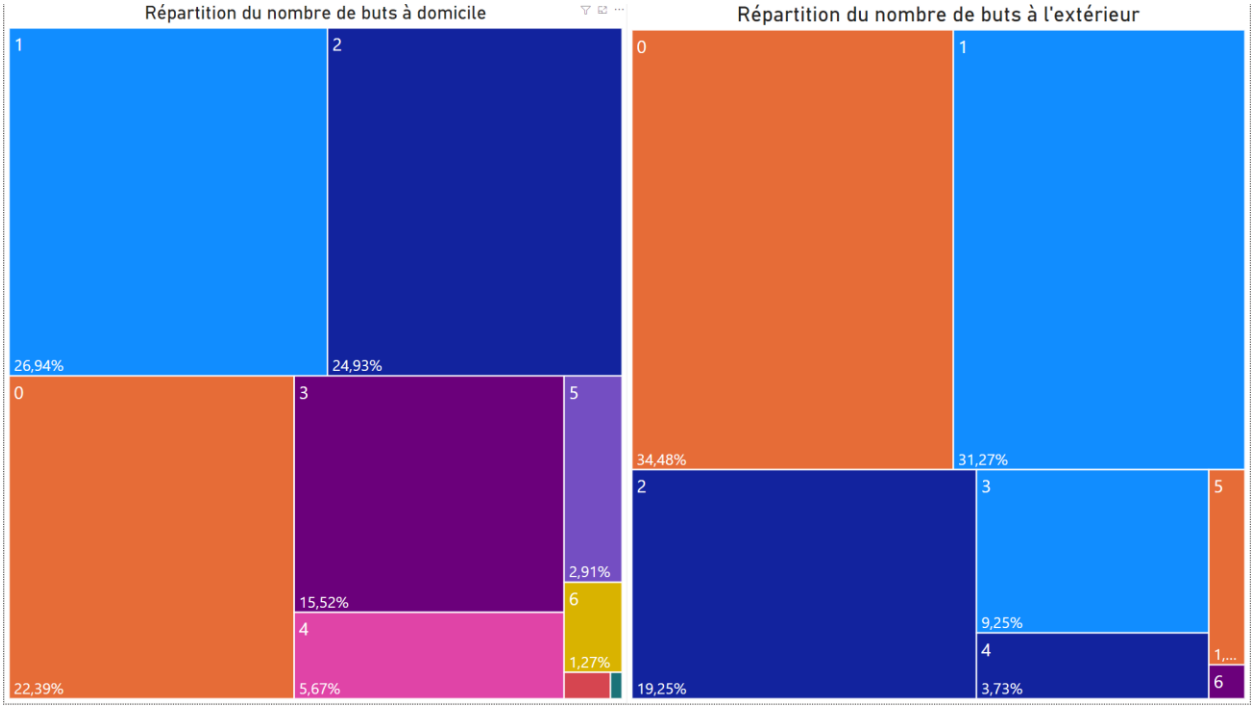


Figure 4: vue Power BI - répartition du nombre de buts marqués à domicile et à l'extérieur

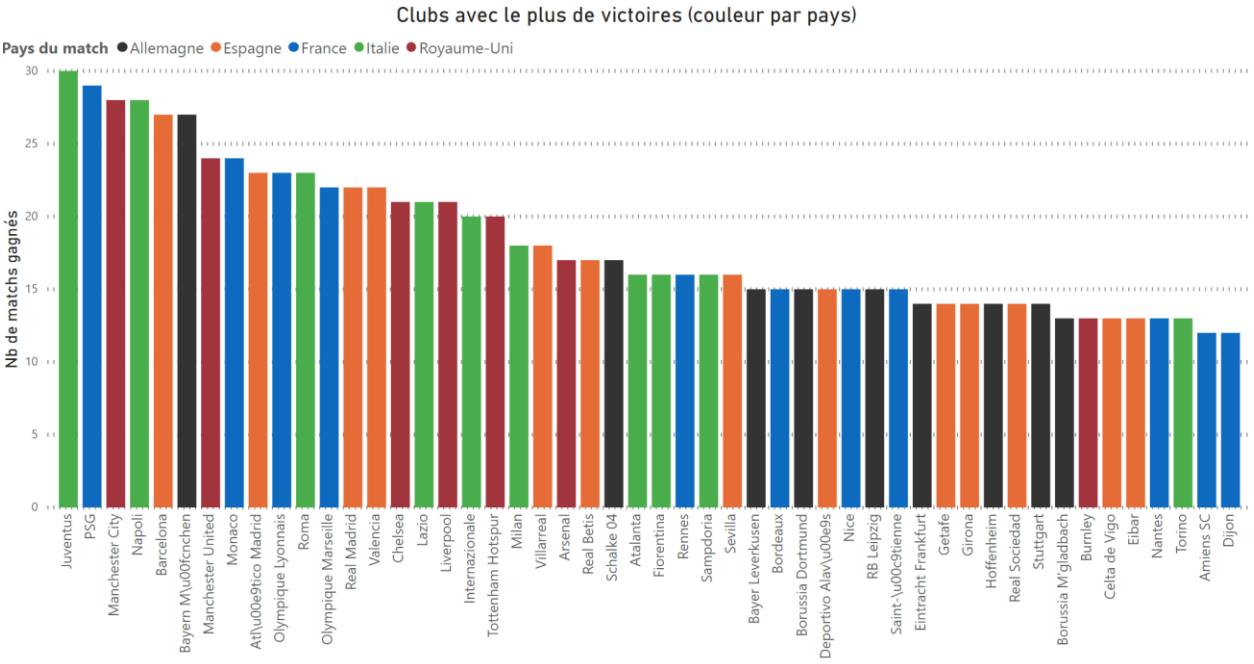


Figure 5: vue Power BI - clubs avec le plus grand nombre de victoires dans la saison, selon leur championnat

Le notebook « Bet-Py_Notations.ipynb » contient un grand nombre de visualisation de données concernant la table match_event selon :

- L'ensemble des events différents selon leur thématique (passes, duel, etc.)
- Une analyse spécifique sur un match par exemple

Sur le Notebook, les différents types d'event ont été étudiés selon les actions possibles selon le thème.

Événement	Passe	Duel	Action autre	Tir	Coup franc	Faute	Arrêt	Hors-jeu	Sortie du gardien	Interruption
Action	Passe simple	Duel attaque au sol	Contact	Tir	Tir du gardien	Faute	Arrêt réflexe		Sortie du gardien	Coup de siflet
	Passe haute	Duel défense au sol	Accélération		Coup de coin	Temps perdu sur faute	Tentative d'arrêt			Ballon en dehors des limites du terrain
	Lancer	Duel perte de balle	Dégagement		Touche	Faute en dehors du terrain				
	Passe intelligente	Duel aérien			Coup franc	Faute de main				
	Passe croisée				Tir de coup franc	Carton en retard				
	Passe de la tête				Coup franc croisé	Protestation				
	Passe à la main				Pénalty	Simulation				
						Faute violente				

Figure 6: différents évènements possibles et les actions associées

Nous pouvons par exemple regarder la distribution des différents events enregistrés dans la base avec plus de la moitié qui concernent des passes (55%) :

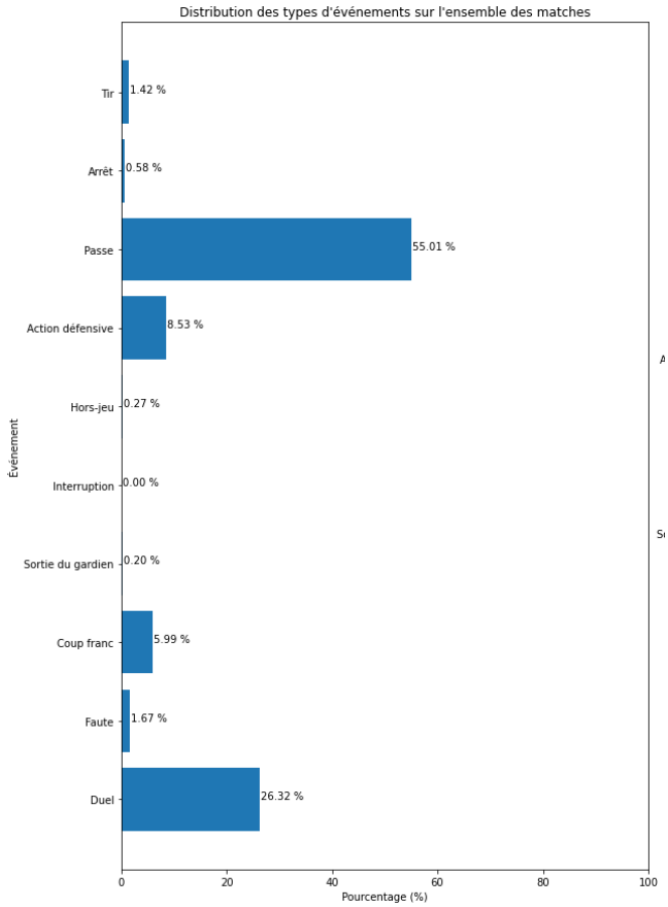


Figure 7: répartition du nombre d'évènements selon leur type dans « match_event »

Le nombre d'évènements selon le poste du joueur peut aussi être une bonne indication. En effet, une équipe qui en proportion a beaucoup plus d'évènements avec ses attaquants doit avoir plus de chance de marquer et donc de gagner le match :

```
Entrée [9]: # Proportion des actions dans laquelle chaque poste est impliqué
            ActionsPie(players_actions, 'position', "Nombre d'actions")
```

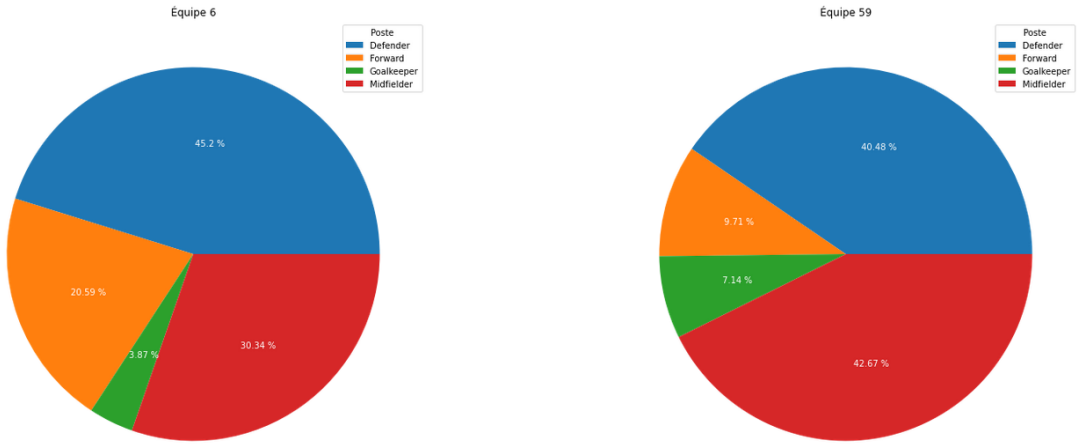


Figure 8: répartition des évènements par type de poste avec 2 équipes en exemple

<u>Compléments :</u>

D'autres vue Power BI ou graphiques Python sont disponibles respectivement dans le fichier « Rapport BI visualisation data.pbix » et le notebook « Bet-Py_Notations.ipynb ». Ces 2 fichiers sont respectivement dans les dossiers « 1 – DataViz » et « 3 – Notebooks » du dossier sous GitHub.

3. Nettoyage, mise en forme et choix des features

A partir des tables d'entrée, des dataframes « match_results » ainsi que « match_infos » ont été créés, et ce en deux temps comme le montre la vue ci-dessous :

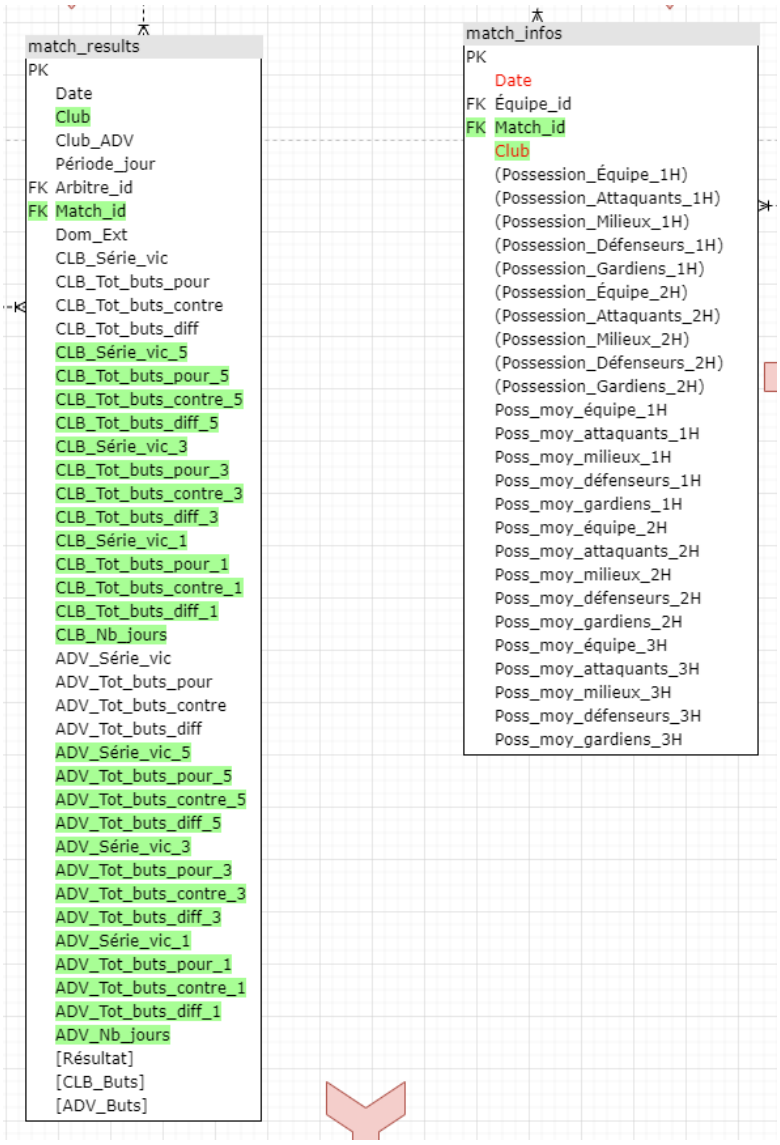


Figure 9: Vue Draw IO des tables match_results et match_infos créées

Première itération complète

L'objectif à ce stade du projet était **d'avoir « rapidement » une première version même incomplète de données afin de pouvoir tester un premier modèle.**

Pour **« être à jeu égal »** et « ne pas tricher » **avec** ce dont disposent **les bookmakers**, nous avons décidé de ne pas **utiliser** toute la base de données mais **uniquement des données des matchs précédents la rencontre où l'on veut prédire le gagnant.**

Lors de cette première itération, les features contenaient les informations « simples » de la table « match » ainsi qu'un grand nombre d'indicateurs contenant de nombreuses informations de statistiques sur les possession des équipes en 1ère et 2nd mi-temps, et ce selon le poste des joueurs ayant la possession (notés dans la table « match_infos »).

Cette première itération a nécessité également du nettoyage avec des clubs ayant différentes syntaxes (Angers et Angers SCO par exemple), des caractères spéciaux donnant également des données d'entrées difficiles à lire ('Saint-\\u00c9tienne' pour Saint-Etienne par exemple) et des lignes sans un club renseigné où nous avons été récupérer l'information dans les calendriers passés de matchs.

Seconde itération

Après parcours d'un état de l'art sur le sujet sur ce type de problème, dont l'article suivant[1], nous avons défini que ce qui pourrait améliorer la performance de notre modèle serait l'obtention d'indicateurs de tendance sur le dernier, les 3 derniers ainsi que les 5 derniers matchs.

Avec notre connaissance également du football, nous avons défini comme élément importants :

- La série de victoire d'affilée sur les 5 derniers matchs
- Le nombre de buts marqués
- Le nombre de buts encaissés
- Le nombre de jours depuis le dernier match (si c'est grand, cela suggère un bon nombre de matchs nuls réalisés dans les derniers matchs)

A ces indicateurs, nous avons rajouté les indicateurs suivants (non décrits dans la vue DrawIO ci-dessous) présentés en écart entre le club évalué et ses adversaires du ou des derniers matchs :

- L'activité des attaquants
- L'activité des défenseurs
- Le nombre de fautes
- Le nombre de passes
- Le nombre de tirs

Tous ces indicateurs ont été calculés pour le club à domicile et celui à l'extérieur et comme précisé plus haut, cela a été réalisé sur le dernier, les 3 derniers et les 5 derniers matchs.

La table complète « match_all » est décrite ci-dessous avec les features retenues en vert :

[1] : https://towardsdatascience.com/machine-learning-algorithms-for-football-prediction-using-statistics-from-brazilian-championship-51b7d4ea0bc8

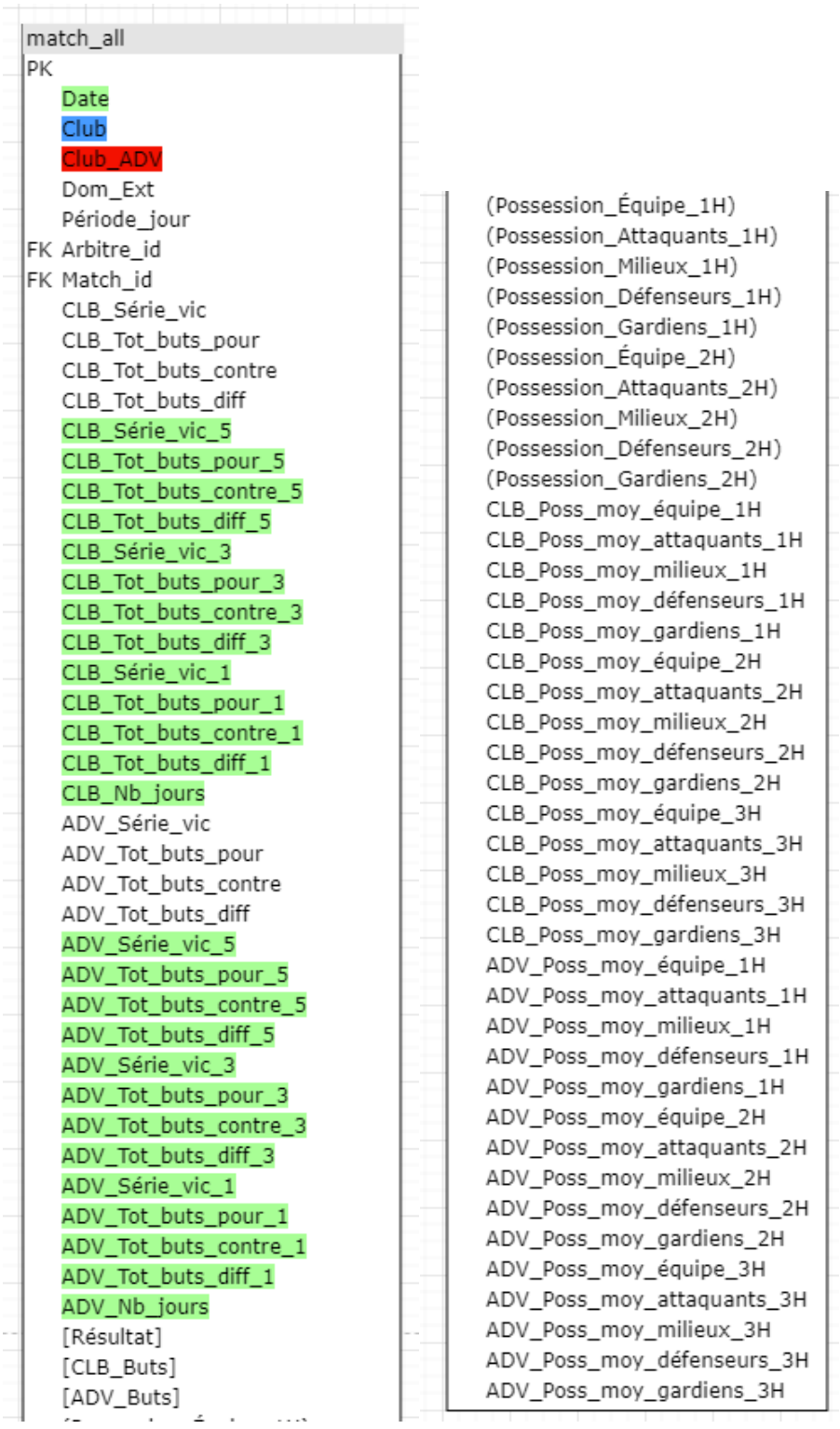


*Figure 10: Table match_all avec les champs en "vert" représentant **une partie** des features retenues pour la 2nd itération*

Définition jeux train / test et standardisation des données

Avec toutes ces données, quelle que soit l'itération (première ou seconde), un découpage en jeu d'entraînement et de test ainsi qu'une étape de standardisation ont été réalisées.

Préparation du Jeux De Données

Entrée [23]:
```
## Sélection du JDD à utiliser
dt     = data.drop(['[Résultat]', '[CLB_Buts]', '[ADV_Buts]'], axis=1)
target = data['[Résultat]']
```

Entrée [24]:
```
## Séparation des données pour entraînement des modèles
#  -> les données sont triées dans l'ordre où les matchs se sont déroulés
X_train, X_test, y_train, y_test = train_test_split(dt, target, test_size=.2, shuffle=False)
```

Entrée [25]:
```
# Changement dimensions suite au commentaire plus bas
y_train=np.ravel(y_train)
y_test=np.ravel(y_test)

# Instanciation scaler et application à X_train SAUF sur le fait d'être à domicile ou à l'extérieur
X_train2=X_train.drop(["Dom_Ext"],axis=1)
X_test2=X_test.drop(["Dom_Ext"],axis=1)
Dom_Ext_train=X_train["Dom_Ext"]
Dom_Ext_test=X_test["Dom_Ext"]
# On garde les index en mémoire
index_train=X_train.index
index_test=X_test.index

scaler = preprocessing.StandardScaler().fit(X_train2)
X_train_scaled=scaler.transform(X_train2)
#Puis standardiser la partie X_test sur le même scaler :
X_test_scaled=scaler.transform(X_test2)

# Puis on vient remettre le champ domicile ou extérieur et les index
X_train_scaled=pd.DataFrame(X_train_scaled)
X_train_scaled.index=index_train
X_train_scaled.insert(0,"Dom_Ext",Dom_Ext_train,True)

X_test_scaled=pd.DataFrame(X_test_scaled)
X_test_scaled.index=index_test
X_test_scaled.insert(0,"Dom_Ext",Dom_Ext_test,True)
X_test_scaled.head()

# Et enfin on remet le nom des colonnes
X_train_scaled.columns=X_train.columns
X_test_scaled.columns=X_test.columns
```

Figure 11: étapes de découpages train/test et standardisation des données

4. Modèle : création, application et évaluation

A partir des ces données complètes, nous avons donc pu attaquer la partie de modélisation.

Une première itération complète

Pour cette première itération, nous avons souhaité tester différents algorithmes :

- SVM avec une grille de recherche (simple avec une combinatoire de 27 possibilités)
- KNN
- Random Forest

A. GridSearch SVM

Entrée [109]:
```
# Test sur des algorithmes unitaires

# Création d'un classificateur SVM
clf=svm.SVC(gamma=0.01,kernel='poly')

parametres={"C":[0.1,1,10],"kernel":['rbf', 'linear','poly'],"gamma":[0.001, 0.1, 0.5]}
grid_clf=model_selection.GridSearchCV(estimator=clf,param_grid=parametres,random_state=123)

grille = grid_clf.fit(X_train_scaled,y_train)
```

```
Les meilleurs paramètres de la grille de recherche sont :  {'C': 10, 'gamma': 0.001, 'kernel': 'rbf'}
```

Entrée [159]:
```
# Application sur les meilleurs paramètres
clf=svm.SVC(gamma=0.001,kernel='rbf',C=10,random_state=123)
clf.fit(X_train_scaled, y_train)
y_pred=clf.predict(X_test_scaled)
print("Score de la prédiction : ","%.3f"%clf.score(X_test_scaled,y_test))
pd.crosstab(y_test, y_pred, rownames=['Classe réelle'], colnames=['Classe prédite'])
```

```
Score de la prédiction :  0.69
```

Avec le modèle SVM, notre meilleure paramétrage donnait un score de 69%.

B. KNN

60]:
```
from sklearn import neighbors
knn=neighbors.KNeighborsClassifier(n_neighbors=7,metric="minkowski")
knn.fit(X_train_scaled, y_train)
y_pred=knn.predict(X_test_scaled)
print("Score de la prédiction : ","%.3f"%knn.score(X_test_scaled,y_test))
pd.crosstab(y_test, y_pred, rownames=['Classe réelle'], colnames=['Classe prédite'])
```

```
Score de la prédiction :  0.626
```

Le modèle des K plus proches voisins donnait quant à lui un score de 63%.

C. Random Forest

```
61]: from sklearn import ensemble
     clf=ensemble.RandomForestClassifier(n_jobs=-1,random_state=123)
     clf.fit(X_train_scaled,y_train)
     y_pred=clf.predict(X_test_scaled)
     print("Score de la prédiction : ","%.3f"%clf.score(X_test_scaled,y_test))
     pd.crosstab(y_test, y_pred, rownames=['Classe réelle'], colnames=['Classe prédite'])
```

```
Score de la prédiction :  0.687
```

Enfin le modèle Random Forest donnait quant à lui un score proche de 69% également.

Avec cette première itération, nous approchions des 70% d'accuracy, ce qui était très encourageant pour ce modèle. En effet, avec uniquement des indicateurs de possession, la performance semblait bonne.

Après vérification des indicateurs d'entrée, nous nous sommes rendus compte que les indicateurs de possession avaient une petite erreur de calcul et surtout nous voulions intégrer un plus grand nombre d'indicateurs qui nous semblaient bien plus important pour aider à la prédiction dans une seconde itération. Une grande partie du travail de ce projet a donc été réalisée afin de les calculer (cf. Nettoyage, mise en forme et choix des features / Seconde itération)

Seconde itération avec nouvelles features

A partir d'un plus grand nombre de données et des modèles plus poussés, nous avons cette fois testé :

- SVM avec grille de recherche bien plus complexe (105 simulations)
- KNN avec également une grille de recherche (51 simulations)
- Random Forest
- Un réseau de neurones « simple » avec deux couches Dense

1. GridSearch SVM

```
]: # Test sur des algorithmes unitaires

   # Création d'un classificateur SVM
   clf=svm.SVC(gamma=0.01,kernel='poly',random_state=123)

   parametres={"C":[0.01,0.1,1,3,5,7,10],"kernel":['rbf', 'linear','poly'],"gamma":[0.001,0.005, 0.1,0.25, 0.5]}
   grid_clf=model_selection.GridSearchCV(estimator=clf,param_grid=parametres,)

   grille = grid_clf.fit(X_train_scaled,y_train)
```

Pour le modèle SVM avec la grille de recherche, le meilleur score était de 66,2%.

2. KNN

```
]: knn=neighbors.KNeighborsClassifier(n_neighbors=7,metric="minkowski")
   knn.fit(X_train_scaled, y_train)
   y_pred=knn.predict(X_test_scaled)
```

```
# Test avec grille de recherche
# Création d'un classificateur KNN
knn=neighbors.KNeighborsClassifier(n_neighbors=7,metric="minkowski")

parametres={"n_neighbors":[4,5,7,10,15,30,50,75,100,250,500,750,1000],"metric":['minkowski','euclidean','manhattan','
grid_knn=model_selection.GridSearchCV(estimator=knn,param_grid=parametres,)

grille = grid_knn.fit(X_train_scaled,y_train)
```

Pour le modèle KNN avec la grille de recherche, le meilleur score était de 65,7%.

3. Random Forest

```
2]: clf=ensemble.RandomForestClassifier(n_jobs=-1,random_state=123)
    clf.fit(X_train_scaled,y_train)
    y_pred=clf.predict(X_test_scaled)
    print("Score de la prédiction : ","%.3f"%clf.score(X_test_scaled,y_test))
    pd.crosstab(y_test, y_pred, rownames=['Classe réelle'], colnames=['Classe prédite'])
```

```
Score de la prédiction :  0.657
```

Classe prédite	0.0	1.0
Classe réelle		
0.0	140	73
1.0	73	140

Pour le modèle Random Forest, le score est de 65,7% également, avec la matrice de confusion présentée ci-dessus.

4. Réseau de neurones

2]:
```
## Modification des vecteurs de validation
y_train_nn = np_utils.to_categorical(y_train, dtype='int')
y_test_nn  = np_utils.to_categorical(y_test, dtype='int')
```

3]:
```
## Préparation du réseau de neuronnes

# Instanciation
clf_nn = Sequential()

# Ajout de la première couche
clf_nn.add(Dense(units=10,
                 input_dim=X_train_scaled.shape[1],
                 kernel_initializer='normal',
                 activation='tanh'))

# Ajout de la seconde couche
clf_nn.add(Dense(units=2,
                 kernel_initializer='normal',
                 activation='softmax'))

# Compilation
clf_nn.compile(loss='categorical_crossentropy',
               optimizer='adam',
               metrics=['accuracy'])
```

4]:
```
## Affichage du réseau de neuronnes
clf_nn.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 10)                380
_________________________________________________________________
dense_1 (Dense)              (None, 2)                 22
=================================================================
Total params: 402
Trainable params: 402
Non-trainable params: 0
```

Le réseau de neurones comporte deux couches Dense et l'entraînement a été réalisé de la manière suivante :

:
```
## Entraînement du réseau de neuronnes
training_history = clf_nn.fit(X_train_scaled, y_train_nn,
                              epochs=10,
                              batch_size=170,
                              validation_split=.2)
```

Malheureusement, les résultats n'étaient pas meilleurs, avec un score à 64% comme le montre le tableau de metrics ainsi que la matrice de confusion ci-dessous.

Classe prédite	0	1
Classe réelle		
0.0	137	76
1.0	76	137

```
]: ## Affichage du rapport d'évaluation
   print(metrics.classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

         0.0       0.64      0.64      0.64       213
         1.0       0.64      0.64      0.64       213

    accuracy                           0.64       426
   macro avg       0.64      0.64      0.64       426
weighted avg       0.64      0.64      0.64       426
```

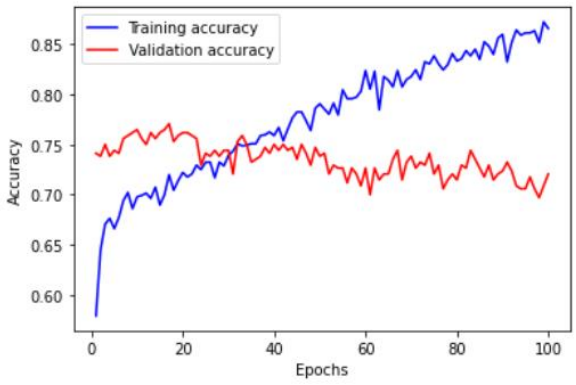
5. Réseau de neuronnes V2

Afin de creuser cette piste de réseau de neurones, nous avons tenter de créer des modèles plus complexes avec plus de couches et des couches de Dropout afin d'éviter au maximum l'overfitting.

```
Model: "sequential_7"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_18 (Dense)             (None, 100)               3800
_________________________________________________________________
dropout_9 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_19 (Dense)             (None, 50)                5050
_________________________________________________________________
dropout_10 (Dropout)         (None, 50)                0
_________________________________________________________________
dense_20 (Dense)             (None, 20)                1020
_________________________________________________________________
dropout_11 (Dropout)         (None, 20)                0
_________________________________________________________________
dense_21 (Dense)             (None, 10)                210
_________________________________________________________________
dense_22 (Dense)             (None, 2)                 22
=================================================================
Total params: 10,102
Trainable params: 10,102
Non-trainable params: 0
_________________________________________________________________
```

Nous avons également souhaité prendre un plus grand nombre d'epochs pour entraîner le modèle.

```
: ## Entraînement du réseau de neuronnes
  training_history2 = clf2_nn.fit(X_train_scaled, y_train_nn,
                                  epochs=100,
                                  batch_size=64,
                                  validation_split=.2)
```



	precision	recall	f1-score	support
0.0	0.64	0.66	0.65	213
1.0	0.65	0.63	0.64	213
accuracy			0.65	426
macro avg	0.65	0.65	0.65	426
weighted avg	0.65	0.65	0.65	426

Avec les différentes itérations faites sur cette seconde itération de neurones, les résultats ne sont que très peu améliorés **avec un score de 65%.**

En conclusion de cette seconde itération avec les nouvelles features qui nous semblaient plus pertinentes ainsi que des tendances sur le derniers, les 3 derniers ou les 5 derniers matchs, nos résultats sont malheureusement moins bons que lors de la première itération.

Conclusion du projet

Ce projet complet nous a permis, comme prévu initialement, de parcourir l'ensemble des étapes d'un projet Data Science. Et comme souvent évoqué la partie « intéressante » de création des algorithmes et d'ajustement des paramètres et hyperparamètres n'est pas la plus longue.

Nous avons pu parcourir toutes ces étapes et obtenir des résultats de prédiction plutôt intéressants (avec un réseau de neurones appris dans les derniers modules), même si ça mériterait bien entendu de les améliorer en modifiant les features et/ou les algorithmes. Nous aurions souhaité aller plus loin sur ce sujet mais l'exploration, le nettoyage et la création des features a été longue et malheureusement un membre de l'équipe projet a dû se retirer rapidement du sujet.

Par rapport à nos 2 itérations, et par rapport au travail d'amélioration, il nous semblerait que la priorité pour améliorer les prédictions serait de passer du temps à ajuster, créer, sélectionner les features intéressantes. En effet, les différents algorithmes et paramétrages associés avaient finalement assez peu de différences. Le « challenge » fixé à la fin de la 1ère itération (même si ambitieux) d'obtenir 75% ne se ferait sans doute pas uniquement avec de meilleurs algorithmes sur ce jeu de données.

Rappel des fichiers à consulter

Pour rappel, voici sous notre dossier GitHub les différents dossiers et notebooks à consulter :

- Visualisation :
  - **1 – DataViz / Rapport BI visualisation data** (fichier Power BI ou pdf associé si vous n'avez pas le logiciel)
  - **3 – Notebooks / Bet-Py_Notations.ipynb**
- Création des données, nettoyage et réalisation des modèles :
  - **3 – Notebooks / Bet-Py_Modélisation.ipynb** (pour arriver à la création de la table « match_all » brute)
  - **3 – Notebooks / Bet-Py_Machines_Learning - Version finale.ipynb** (pour créer et sélectionner certaines features, faire la standardisation des données ainsi que les différents modèles et résultats associés)