



IMT Mines Alès
École Mines-Télécom

Avril 2023

RAPPORT DU PROJET MACHINE LEARNING



SOMMAIRE

01	Introduction	P3
02	Problématique	P4
03	Notre jeu de données	P5 - P7
04	Démarche & Code	P8 - P11
05	Résultats & critiques	P12 - P14
06	Conclusion	P15
07	Appendices	P16 - P21

INTRODUCTION

Depuis des décennies, le football est considéré comme l'un des sports les plus populaires et les plus appréciés dans le monde entier. Chaque année, des milliers de matchs sont disputés dans les championnats nationaux, internationaux et continentaux, et des millions de supporters suivent passionnément les performances de leurs équipes préférées. Avec l'évolution de la technologie et l'émergence de l'intelligence artificielle, les prévisions de scores de match de football sont devenues un sujet d'intérêt croissant pour les amateurs de football et les parieurs sportifs.

L'utilisation de l'intelligence artificielle et du machine learning pour prédire les résultats de matchs de football est une pratique relativement nouvelle, mais qui a connu une croissance rapide au cours des dernières années. Les modèles de prédiction utilisent des algorithmes sophistiqués et des techniques de machine learning pour analyser les données historiques, les statistiques des joueurs et les performances des équipes, afin de prédire avec précision les scores des matchs futurs.

Dans notre cas, on souhaite prédire le résultat d'un match de foot entre l'équipe 1 jouant à domicile et l'équipe 2 jouant à l'extérieur.

En prenant comme référentiel l'équipe 1, 3 cas sont alors envisageables :

- Victoire équipe 1
- Match nul
- Victoire équipe 2



PROBLÉMATIQUE



Afin d'établir son pronostic pour un match de football, plusieurs critères reviennent souvent. Les plus importants étant la différence de buts des deux équipes, le nombre de tirs réalisés et concédés, etc.

Par ailleurs, les cotes données par les bookmakers pour les matchs se révèlent aussi être un fort indicateur. En effet, une côte de victoire à domicile proche de 1 indiquerait une probabilité plus grande de gagner pour cette équipe. On peut ainsi s'aider des analyses faites au préalable par les bookmakers.

Dans notre cas, on dispose d'un jeu de données reprenant plusieurs critères pour chaque équipe, chaque match... La quantité de données étant importante, les méthodes classiques d'apprentissage s'apparentent naturellement au cas de notre étude. **D'où notre problématique :**

Comment prédire le résultat d'un match de foot en utilisant des méthodes de Machine Learning ?

NOTRE JEU DE DONNÉES

Notre jeu de données original est tiré du site web "<https://www.football-data.co.uk>" qui recense les résultats et les caractéristiques des matchs de foot de quasiment tous les championnats du monde depuis le début des années 2000. Pour entraîner nos différents modèles, nous avons décidé de traiter les jeux de données des **5 grands championnats (France, Espagne, Italie, Allemagne, Angleterre)** de la saison 2000/2001 à la saison 2021/2022. Un échantillon des données que nous traitons se présente de la manière suivante :

1	Div,Date,HomeTeam,AwayTeam,FTHG,FTAG,FTR,HTHG,HTAG,HTR,GBH,GBD,GBA,IWH,IWD,IWA,LBH,LBD,LBA,SBH,SBD,SBA,WHH,WHD,WHA
2	F1,28/07/00,Marseille,Troyes,3,1,H,2,1,H,1.65,3.3,4.3,1.45,3.5,5,,,1.53,3.5,5.5,1.45,3.5,6
3	F1,28/07/00,Paris SG,Strasbourg,3,1,H,1,1,D,1.6,3.4,4.6,1.35,3.6,6.5,,,1.4,3.75,7,1.4,3.7,6.5
4	F1,29/07/00,Auxerre,Sedan,0,1,A,0,1,A,1.7,3.3,3.9,1.7,3.1,3.8,,,1.7,3.25,4.5,1.65,3.25,4.7
5	F1,29/07/00,Bordeaux,Metz,1,1,D,1,0,H,1.65,3.3,4.3,1.55,3.3,4.5,,,1.57,3.4,5.25,1.5,3.4,6
6	F1,29/07/00,Guingamp,St Etienne,2,2,D,2,1,H,2.25,3,2.75,2,2,2.8,2.8,,,2.5,3.2,2.2,9,3,2.25
7	F1,29/07/00,Lille,Monaco,1,1,D,0,1,A,2.75,3,2.25,2.4,2.9,2.4,,,3.25,3.2,2,2.9,3,2.25
8	F1,29/07/00,Lyon,Rennes,2,2,D,0,2,A,1.6,3.4,4.6,1.5,3.4,4.7,,,1.44,3.6,6.5,1.55,3.5,5.3
9	F1,29/07/00,Nantes,Lens,0,2,A,0,0,D,2.15,3,3,2,2.9,3,,,2.25,3.2,2.75,2,1,3.1,3.1

Image 1 - Capture d'écran d'un échantillon du jeu de données "dataset.csv"

L'échantillon présenté est un extrait du dataset du championnat français de la saison 2000/2001. À chaque match sont associées 25 types de données différentes. On énumère dans ce rapport les acronymes des données utilisées et leur signification (les autres colonnes portent un nom explicite de la donnée traitée) :

- FTR = Full Time Result (H=Home Win, D=Draw, A=Away Win)
- FTHG = Full Time Home Team Goals
- FTAG = Full Time Away Team Goals
- HS = Home team Shoot
- AS = Away team Shoot
- HST = Home team Shoot on Target
- AST = Away team Shoot on Target
- HC = Home team Corners
- AC = Away team Corners
- HF = Home team Fouls committed
- AF = Away team Fouls committed
- HY = Home team Yellow cards
- AY = Away team Yellow cards
- HR = Home team Red cards
- AR = Away team Red cards
- WHH = William Hill "Home win" odds
- WHD = William Hill "Draw" odds
- WHA = William Hill "Away win" odds

Par ailleurs, il a été nécessaire de nettoyer ce jeu de données puisque certaines lignes comportaient soit des erreurs soit des informations manquantes. Finalement, on réussit à recenser les données d'un peu plus de 30.000 matchs au cours des 20 dernières années. Ainsi, à partir de ce dataset, on souhaite maintenant construire un nouveau dataset plus conséquent et plus optimal que nous utiliserons dans nos jeux d'entraînement et de test.

NOTRE JEU DE DONNÉES

Une fois le dataset initial implémenté, il a été question de récupérer les données les plus intéressantes mais aussi de calculer celles qui ne sont pas présentes naturellement dedans (par exemple le cumul des buts au cours d'une saison par équipe, des victoires / défaites au cours d'une saison par équipe, des fautes commises au cours d'une saison par équipe...). De plus, les données de certaines saisons ne sont pas accessibles selon les pays. Il a fallu donc adapter à quels dataframes les fonctions présentent dans notre pre-processing seront appliqués.

On propose dans le rapport l'exemple de la fonction "foul_made" qui prend en entrée un dataframe (plus précisément les colonnes relatives aux fautes dans notre dataset) et renvoie un dataframe du cumul des fautes réalisées par journée, par équipe. On remarque aussi la présence de la variable COVID en entrée de cette fonction (et de toutes nos autres fonctions par ailleurs) qui a été un facteur perturbateur pour la saison 2019/2020. En effet, la taille du dataframe des fautes cumulées par journée n'est pas la même en France car le championnat a été arrêté à la 27ème journée sur 38.

```
def foul_made(df, covid):
    # Création du dictionnaire des équipes de la saison
    teams = {}
    for i in df['HomeTeam'].unique():
        teams[i] = []
    # Journée par journée, on ajoute le nombre de fautes réalisées
    for _, row in df.iterrows():
        foul_home = row['HF']
        foul_away = row['AF']
        teams[row['HomeTeam']].append(foul_home)
        teams[row['AwayTeam']].append(foul_away)
    # On ajoute ces données dans un dataframe équipe par équipe, journée
    colonnes = list(range(1, 2*(len(teams)-1)+1))
    if covid:
        colonnes = list(range(1, 28))
    df_foul = pd.DataFrame(data = teams, index=colonnes).T
    df_foul.insert(0, 0, 0)
    # On cumule le nombre de fautes
    for i in colonnes:
        df_foul[i] = df_foul[i] + df_foul[i-1]
    return df_foul
```

Image 2 - Fonction "foul_made" qui compte le nombre de fautes cumulées réalisées par équipe, par journée

Ici, grâce à la fonction "foul_made" & "foul_conc" (qui agit de la même manière que la fonction "foul_made" mais pour les fautes subis et non plus réalisés par journée, par équipe), on réussit à créer 4 nouvelles métriques que sont "HomeFoulMade", "HomeFoulConceded", "AwayFoulMade" et enfin "AwayFoulConceded" qui nous seront utiles comme features pour nos modèles. Ce fonctionnement a été appliqué aussi pour les buts, les tirs, les cartons, les corners et un système de points en fonction du résultat du match a été ajouté créant ainsi encore plus de nouvelle features utiles pour nos différents modèles.

NOTRE JEU DE DONNÉES

Finalement, on passe de 25 colonnes dans le dataset initial à 42 colonnes dans notre dataset final. Toutes les colonnes rajoutées portent des noms explicites de ce qu'elles contiennent. Ainsi, grâce à cette réorganisation des données, on peut dresser certains résultats intéressants concernant notre dataset.

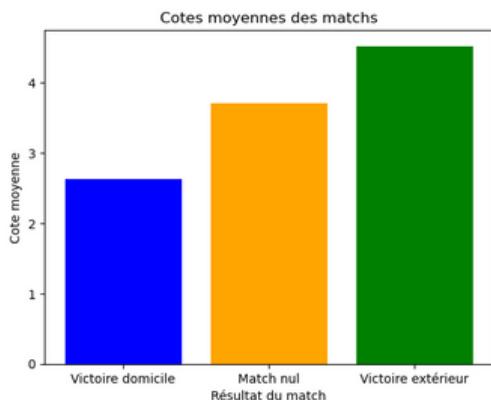


Image 3 - Côte moyennes des matchs pour les 5 grands championnats des saisons 00/01 à 21/22

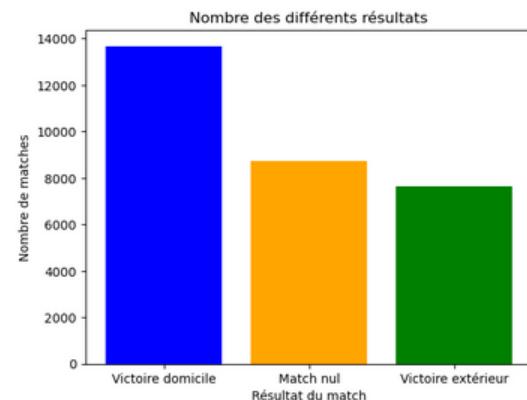


Image 4 - Résultats des matchs pour les 5 grands championnats des saisons 00/01 à 21/22

On observe tout d'abord que, selon les côtes, une victoire à domicile est beaucoup plus probable qu'un match nul ou encore qu'une victoire à l'extérieur, ce qui devrait probablement influencer les résultats de nos modèles. On note aussi que dans notre dataset, les victoires à domicile sont bien plus présentes que les autres résultats, il faudra donc sous-échantillonner préalablement le dataset pour évaluer nos modèles.

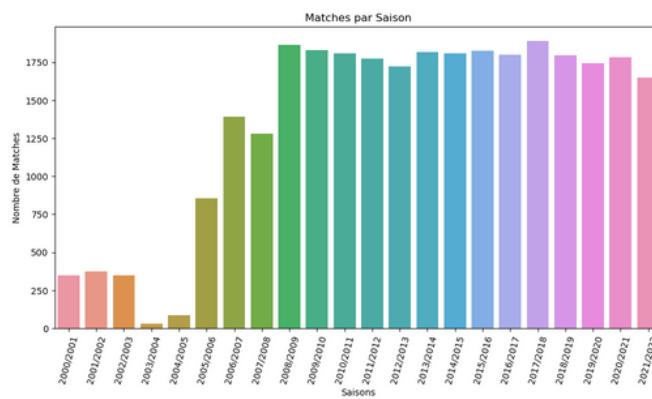


Image 5 - Matches par saison pour les 5 grands championnats des saisons 00/01 à 21/22

On observe ici qu'il manque des données sur certaines saisons. C'est à cause de problèmes de format sur les certains datasets. On a donc retiré ces données.

DEMARCHE & CODE

Pour répondre à notre problème de classification, nous avons utilisés 7 méthodes différentes. Pour chaque méthode, on expliquera alors la méthode et le code utilisé.

Régression Logistique

Cette méthode est en général utilisé lorsque le problème traite d'une problématique de classification binaire. Ici, étant donné les 3 classes à prédire, on utilise une approche OvR (One vs Rest). Chaque modèle est entraîné pour identifier une classe cible en la différenciant des autres.

Dans notre cas nos 3 méthodes sont :

- Victoire contre Nul et Défaite
- Nul contre Victoire et Défaite
- Défaite contre Nul et Victoire

Chaque modèle produit une probabilité pour sa classe et c'est finalement la classe avec la probabilité la plus élevée qui est choisie comme prédiction finale.

Notre code se décompose en les étapes suivante :

- On crée un objet de régression logistique, `clf_RL`, en spécifiant un état aléatoire pour la reproductibilité des résultats.
- On ajuste (entraîne) le modèle de régression logistique en utilisant les données d'apprentissage (`X_train` et `y_train`).
- On effectue des prédictions sur les données de test (`X_test`) à l'aide du modèle entraîné, et les résultats sont stockés dans `y_pred_RL`.
- On évalue la performance du modèle en calculant la précision (accuracy) à l'aide de la méthode `score`. La précision est déterminée en comparant les prédictions du modèle (`y_pred_RL`) aux véritables valeurs de classe (`y_test`).

Pour ce qui est de l'approche OvR, celle-ci est implémentée par défaut avec **LogisticRegression** de la bibliothèque scikit-learn.

DEMARCHE & CODE

Réseaux Neuronaux

Nous avons choisi d'implémenter un réseau de neurones afin de tester la précision de ce type de modèle sur nos données. Voici le résumé de celui-ci :

Layer (type)	Output Shape	Param #
dense (Dense)	(None, 256)	10752
dropout (Dropout)	(None, 256)	0
dense_1 (Dense)	(None, 128)	32896
dropout_1 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 64)	8256
dropout_2 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 32)	2080
dropout_3 (Dropout)	(None, 32)	0
dense_4 (Dense)	(None, 3)	99

La couche d'entrée est composée de 256 neurones possède une fonction d'activation ReLu et est suivie d'un Dropout de 0,5 pour ne garder que 50% des neurones de cette couche durant l'entraînement, cela permettra de prévenir l'overfitting.

Les 3 couches cachées sont composées de 128, 64 et 32 neurones et ont pour fonction d'activation la tangente hyperbolique car cela améliore les résultats. Elles sont aussi toutes suivies d'un dropout de 0,2.

La couche de sortie contient 3 neurones car on a 3 classes et une fonction d'activation softmax car c'est la couche de classification.

Pour prévenir l'overfitting, on applique la L2 Regularization aux 4 premières couches.

On choisit comme fonction de calcul d'erreur la Sparse Categorical Cross Entropy car on est sur un problème de classification à 3 classes.

On modifie aussi le learning rate à 0,001 pour optimiser l'entraînement. De plus, on trace les courbes de calcul d'erreur des datasets d'entraînement et de validation et on détermine que le nombre d'epochs optimal est d'environ 50.

DEMARCHE & CODE

Arbre Décisionnel

La méthode étant détaillé dans l'appendice, on s'attardera uniquement sur le code.

clf_DT = DecisionTreeClassifier(random_state=1):

Crée un objet d'arbre de décision nommé **clf_DT** avec un état aléatoire fixé à 1 pour assurer la reproductibilité des résultats.

clf_DT.fit(X_train, y_train):

Entraîne le modèle d'arbre de décision en utilisant les données d'entraînement (**X_train**, **y_train**). **X_train** contient les caractéristiques (variables indépendantes), et **y_train** contient les étiquettes de classe (variables dépendantes).

y_pred_DT = clf_DT.predict(X_test):

Utilise le modèle entraîné pour prédire les étiquettes de classe des données de test (**X_test**) et stocke les prédictions dans **y_pred_DT**.

accuracy_DT = clf_DT.score(X_test, y_test):

Calcule et stocke l'exactitude (accuracy) du modèle d'arbre de décision en comparant les prédictions (**y_pred_DT**) aux véritables étiquettes de classe (**y_test**) des données de test.

KNN

Pour ce qui est de l'algorithme, Le k plus proches voisins (KNN) est un algorithme d'apprentissage supervisé utilisé pour la classification et la régression. Son principe repose sur l'idée que des objets similaires se trouvent généralement à proximité les uns des autres dans l'espace des caractéristiques.

Pour la classification, KNN fonctionne en trouvant les k instances les plus proches (voisins) de l'objet dont la classe doit être prédite. La classe majoritaire parmi ces voisins est ensuite attribuée à l'objet en question. Le paramètre k détermine le nombre de voisins à prendre en compte pour la prédiction. Ici on choisit de fixer le nombre de voisins à considérer à 8.

Comme pour les précédents codes, on a 4 différentes étapes, avec la création de l'objet **clf_KNN**, l'entraînement du modèle, les prédictions et l'évaluation du modèle.

DEMARCHE & CODE

Random Forest

La méthode étant détaillé dans l'appendice, on s'attardera uniquement sur le code.

De même on a ici, création du modèle, entraînement, prédiction avec le jeu de test et évaluation du modèle.

SVM

La méthode Support Vector Machine (SVM) est un algorithme d'apprentissage supervisé utilisé principalement pour la classification et la régression.

Après avoir représenté les données en fonction de leurs caractéristiques, l'objectif est de trouver un hyperplan qui sépare les deux classes avec la plus grande marge possible. La marge est définie comme la distance entre l'hyperplan et les points les plus proches de chaque classe, appelés vecteurs de support. L'hyperplan optimal est celui qui maximise cette marge.

Dans le cas de notre problème de classification 3 classes, on utilise l'approche OvO (One vs One) naturellement implémenté par **SVC** avec scikit-learn.

On choisit d'utiliser le noyau RBF (Radial Basis Function) très souvent utilisé pour la méthode SVM. Pour ce qui est de l'approche One vs One, un classificateur binaire est entraîné pour chaque paire de classes, soit un total de 3 classificateurs. Lors de la prédiction, l'algorithme choisit la classe qui remporte le plus grand nombre de "duels".

XGBoost

XGBoost (eXtreme Gradient Boosting) est un algorithme d'apprentissage supervisé basé sur l'amélioration du gradient. Il est utilisé pour les problèmes de classification, de régression et de classement.

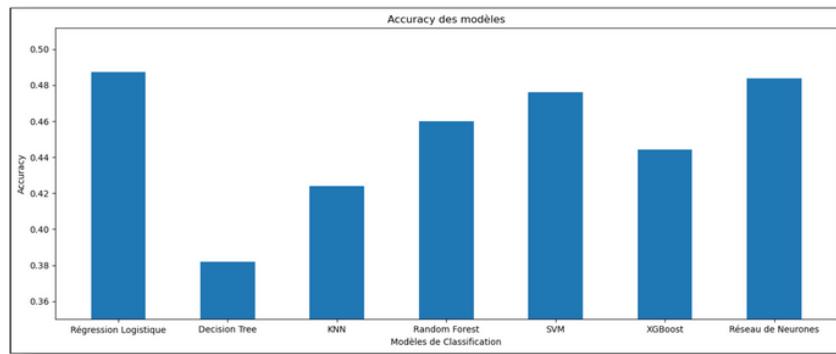
Voici les principales caractéristiques de la méthode XGBoost:

XGBoost utilise des arbres de décision comme apprenants faibles. Ces arbres sont construits de manière itérative, en ajoutant un nouvel arbre à chaque étape pour corriger les erreurs commises par les arbres précédents.

L'optimisation est réalisée via la minimisation d'une fonction de perte différentiable en mettant à jour les prédictions à chaque itération en utilisant le gradient de la fonction de perte. Cela permet à l'algorithme d'optimiser la performance du modèle en ajustant les poids des apprenants faibles. De plus, XGBoost introduit une régularisation dans la fonction de perte pour contrôler la complexité du modèle. La régularisation aide à éviter l'overfitting en pénalisant les modèles trop complexes. Le code suivant a été utilisé, il comprend les mêmes étapes qu'auparavant.

RÉSULTATS & CRITIQUES

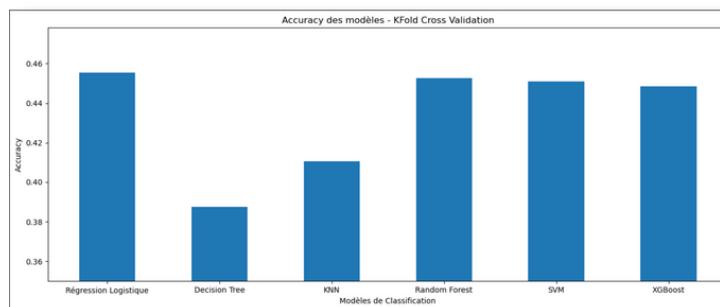
Une fois les 7 méthodes implémentés, on s'intéresse à la précision de chacun des modèles. Voici un premier graphique nous permettant de visualiser cela. On observe que la régression logistique est le modèle le plus performant ici avec un score d'accuracy d'environ 48.734. Le réseau de neurones obtient lui un score d'environ 48.384.



Afin d'évaluer nos modèles de manière générale, on choisit d'effectuer une validation croisée à 5 plis. Les étapes sont les suivantes :

1. Les données sont divisées en 5 sous-ensembles de taille approximativement égale.
2. Le modèle est entraîné et évalué 5 fois, à chaque fois en utilisant un pli différent comme ensemble de test et les 4 autres plis comme ensemble d'entraînement.
3. Les performances du modèle sont calculées en moyennant les scores de performance.

L'objectif de la validation croisée ici est d'obtenir une estimation plus fiable et moins biaisée de la performance du modèle. En utilisant plusieurs découpages différents, on réduit la dépendance des résultats par rapport à un découpage particulier de nos données : dans notre cas donc le découpage 80/20 avec un jeu de test unique. Après évaluation, voici le second graphique obtenu :

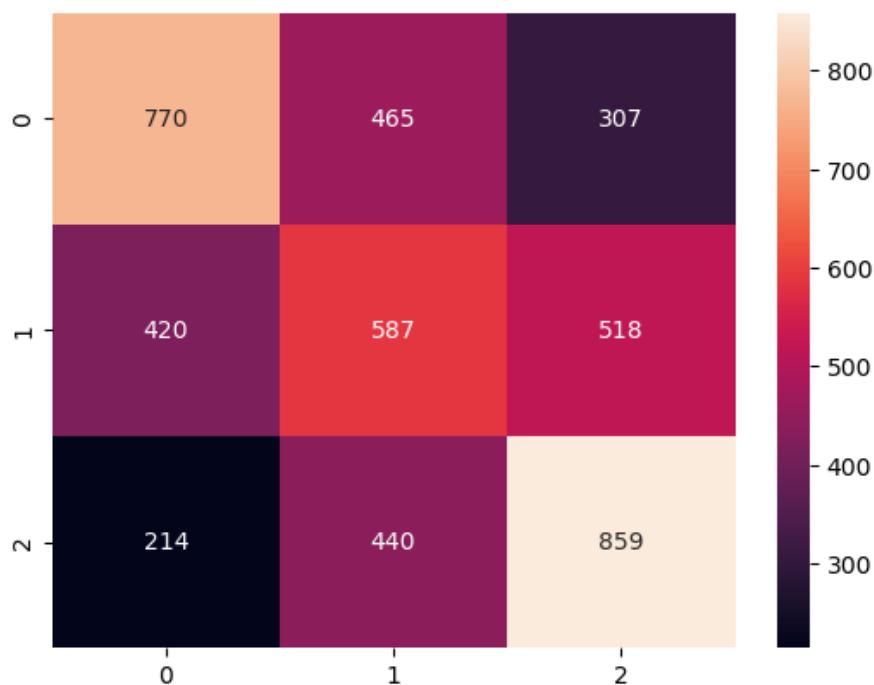


On remarque alors, qu'avec une validation croisée, la précision du modèle de Régression Logistique est diminuée et que la plupart des méthodes convergent se retrouvent autour d'un score de 0,45. Nous n'avons pas réussi à appliquer la validation croisée au réseau de neurones car cela faisait planter tensorflow. Nous n'avons donc pas de résultats mais le code nécessaire est quand même présent en commentaires sur le notebook, vous pourrez donc l'essayer.

RÉSULTATS & CRITIQUES

Passons à l'analyse d'une matrice de confusion, celle du réseau de neurones, qui obtient de relativement bons résultats. Précisons tout d'abord le nom des classes :

- 0 : Victoire équipe extérieur
- 1 : Match nul
- 2 : Victoire équipe à domicile



Si un modèle est fonctionnel, on doit obtenir les plus grandes valeurs dans la diagonale de la matrice. En effet, cela signifie que le modèle a bien associé les classes originales aux classes prédites. On observe que le modèle est plutôt fonctionnel sauf pour prédire des matchs nuls (1). On observe que, pour les matchs nuls, ces derniers sont presque autant prédits que les victoires de l'équipe à domicile. C'est la difficulté de notre modèle, prédire des matchs nuls. On note que le modèle se trompe beaucoup moins entre des victoires et des défaites, comme en témoignent les cases en bas à gauche et en haut à droite. Il y a relativement peu de victoires domicile prédites comme victoires extérieur et inversement.

Une solution pour améliorer l'accuracy de notre modèle aurait été de supprimer les matchs nuls.

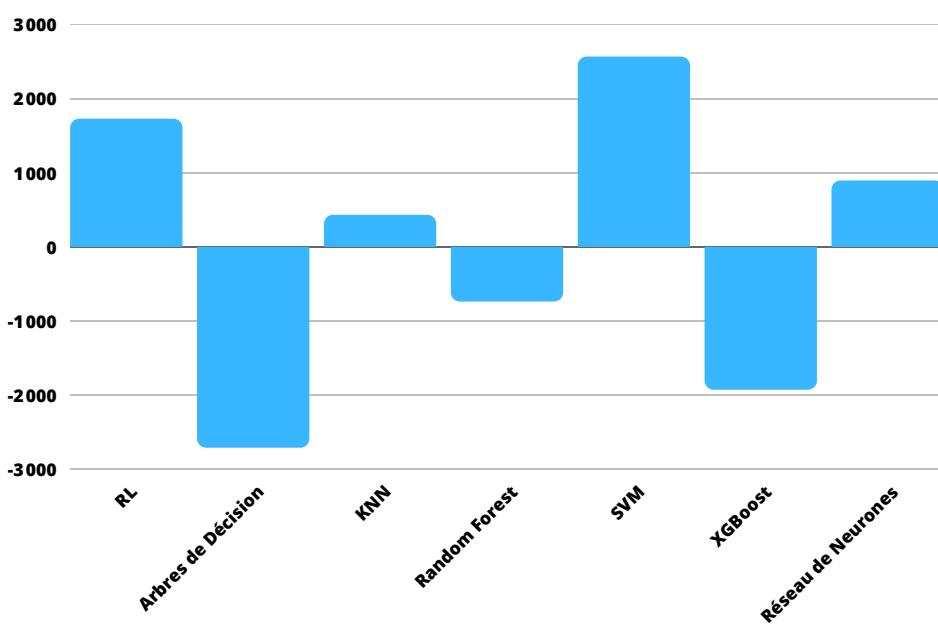
RÉSULTATS & CRITIQUES

Un gain financier ?

Afin de rendre notre analyse plus concrète, on choisit de calculer le gain financier qui pouvait être réalisé dans le cas où un parieur avait accès à chacun de nos modèles et avait parié sur chacun des matchs en suivant les prédictions des modèles. On considère ici chaque match disponible dans notre jeu de test.

On considère alors pour chacun des modèles que le parieur mise la somme de **10 €**. Le gain est alors la mise empochée - moins la mise de départ ce qui correspond au bénéfice.

Pour chacun des modèles, voici les gains envisageables en euros :



On peut ainsi remarquer que :

- Bien que le SVM n'est pas obtenu le score le plus élevé, le gain réalisé serait plus élevé en se basant sur ce modèle. Cela signifie alors que les issues prédites par ce modèle avaient parfois moins de probabilité de se réaliser (côtes plus élevées) que les issues prédites par la méthode de Régression Logistique. Cela reste quand même à nuancer étant donné les scores très proches entre ces deux modèles.
- À l'inverse, la perte envisagée en suivant le modèle XGBoost est plus élevée que celle qu'on pourrait envisager en suivant le modèle Random Forest alors que sa précision est à priori plus élevée. On pourrait alors en déduire que les prédictions réalisées par XGBoost sont souvent celles avec les côtes les plus faibles et donc les plus "évidentes".

CONCLUSION

En conclusion, l'utilisation de l'intelligence artificielle et du machine learning pour prédire les résultats de matchs de football est une avancée majeure dans le domaine des paris sportifs et de l'analyse de données.

Grâce à nos différents modèles de prédiction, un parieur lambda peut désormais prendre des décisions un peu plus éclairées en matière de paris. Cependant, il convient de noter que ces modèles (dont les nôtres) ne sont pas infaillibles et que les résultats des matchs peuvent être influencés par de nombreux autres facteurs imprévisibles tels que la forme des joueurs, les blessures ou encore l'implication des supporters. Par conséquent, il est important de ne pas se fier uniquement aux prédictions de ce genre de modèles, mais de prendre en compte d'autres facteurs avant de prendre une décision de pari.



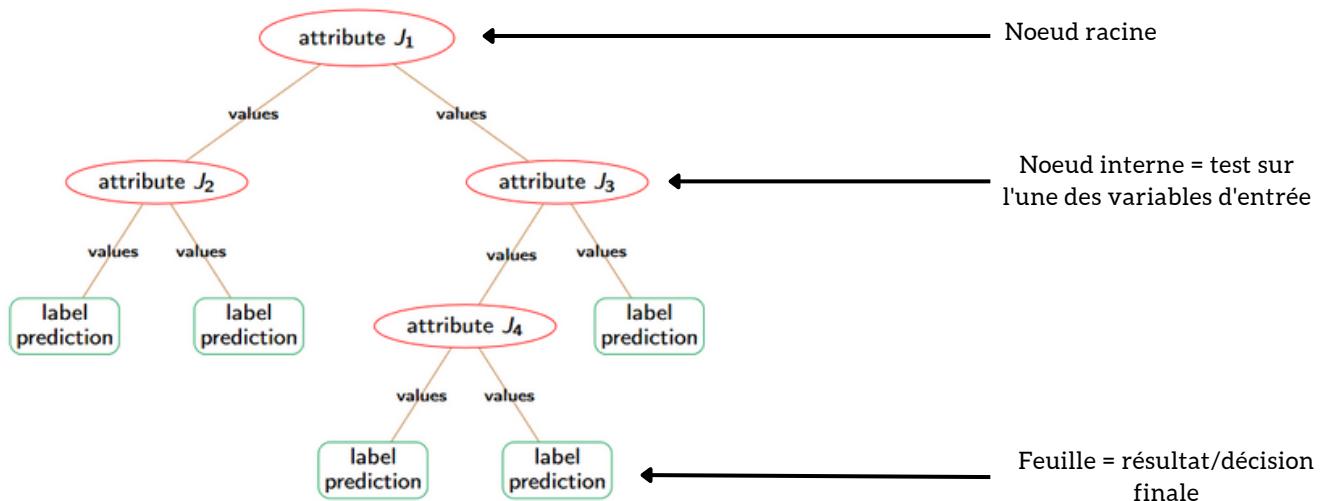
En somme, l'utilisation de l'IA pour prédire les résultats de matchs de football est une technologie prometteuse qui peut aider les parieurs à prendre des décisions plus informées et à améliorer leur taux de réussite.

APPENDICE 1: ARBRES DE DÉCISIONS

Type de problèmes couverts: Apprentissage supervisé --> Régression et Classification

Modèle de type "boîte blanche": résultats de la décision transparents et faciles à comprendre (on peut suivre la séquence décisionnelle)

Représentation:

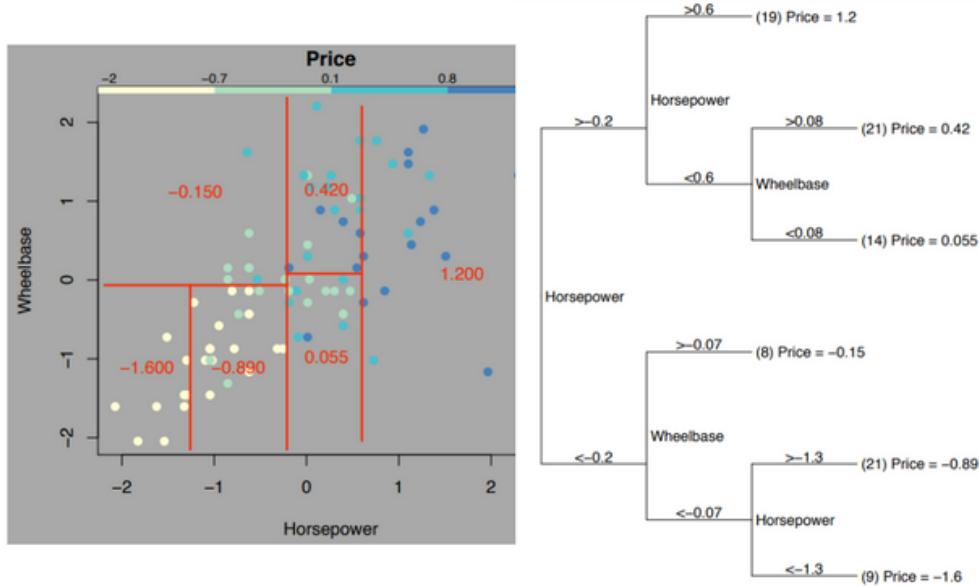


Chaque **noeud interne** de l'arbre représente une **variable d'entrée**, qui est testée à un certain seuil. Si la variable testée est catégorielle, on aura une branche par classes de l'attribut, si elle est continue, on aura une branche par résultat du test (True/False).

Principes de construction d'un arbre:

- principe de '**top-down induction**': l'arbre est construit par partition récursive de chaque noeud en fonction de la valeur de l'attribut testé à chaque itération
- principe de **pureté**: la sélection de la variable d'entrée (ou attribut) qui maximise la séparation entre les différentes classes ou les différents groupes de valeurs est effectuée à l'aide d'un critère de qualité qui mesure la pureté des feuilles (sous-ensembles créés par la partition). Les critères les plus couramment utilisés sont l'entropie de Shannon, le gain d'information (utilisé par les algorithmes ID3 et C4.5) ou le coefficient de Gini (utilisé par l'algorithme CART)

APPENDICE 1: ARBRES DE DÉCISIONS



Sur cette image est illustrée le principe de partitionnement récursif qui vise à **maximise** la séparation entre les différentes classes.

Classification: la pureté correspond à l'homogénéité des labels de classes et est mesurée à l'aide d'indicateurs comme le **coefficient de Gini** ou l'**entropie de Shannon**

- ▶ **CART** → Gini impurity : $i(t_h) = \sum_{k=1}^K p_k(1 - p_k)$ whith
- ▶ **ID3, C4.5** → Shanon entropy : $i(t_h) = - \sum_{k=1}^K p_k \log_2(p_k)$ $p_k = P(Y = \omega_k | t_h)$

avec:

Class label $Y \in \Omega = \{\omega_1, \dots, \omega_K\}$ ($\in \mathbb{R}^K$ for regression)

Tree

$$\mathcal{P}_H = \{t_1, \dots, t_H\} \text{ and } \pi_h = P(t_h) \approx \frac{|t_h|}{N} \text{ with } |t_h| = \#\{i : x_i \in t_h\}$$

Régression: la pureté correspond à une basse variance des labels de classe et est mesurée avec l'indicateur suivant:

$$i(t_h) = \widehat{\text{Var}}(Y|t_h) = \frac{1}{|t_h|} \sum_{x_i \in t_h} (y_i - \widehat{E}(Y|t_h))^2 \text{ with } \widehat{E}(Y|t_h) = \frac{1}{|t_h|} \sum_{x_i \in t_h} y_i$$

APPENDICE 1: ARBRES DE DÉCISIONS

Taille de l'arbre: La taille de l'arbre doit être déterminée de manière à trouver le **bon compromis** entre la **performance** et la **complexité** du modèle.

Il est donc important de choisir la taille optimale de l'arbre en utilisant des techniques comme la validation croisée et la régularisation pour éviter le surapprentissage. Les méthodes de régularisation consistent à limiter la profondeur de l'arbre, le nombre minimum d'exemples nécessaires pour diviser un nœud, ou le nombre minimum d'exemples requis pour être présent dans une feuille.

Eviter l'overfitting: il existe des techniques d'élagage (régularisation) qui sont des solutions algorithmiques permettant de tenter d'éviter au maximum le surapprentissage des modèles.

-Le **pré-élagage** consiste à stopper la construction de l'arbre avant que le modèle ne soit trop complexe, en utilisant des critères tels que la taille minimale d'un groupe ou l'homogénéité d'un sous-ensemble.

-Le **post-élagage**, quant à lui, consiste à construire un arbre complet puis à le réduire en utilisant un ensemble de données distinct, afin d'optimiser les performances du modèle.

Pré-élagage: Voici les critères de pré-élagages

For all leaves $\{t_h\}_{h=1,\dots,H}$ and their potential children :

- ▶ **leaves purity** : $\exists k \in \{1, \dots, K\} : p_k = 1$
- ▶ **leaves and children sizes** : $|t_h| \leq minLeafSize$
- ▶ **leaves and children weights** : $\pi_h = \frac{|t_h|}{t_0} \leq minLeafProba$
- ▶ **leaves number** : $H \geq maxNumberLeaves$
- ▶ **tree depth** : $depth(\mathcal{P}_H) \geq maxDepth$
- ▶ **purity gain** : $\Delta i \leq minPurityGain$

Post-élagage: consiste à trouver un sous-arbre avec un meilleur **compromis qualité prédictive/complexité** du modèle. Pour évaluer cela, on peut utiliser des métriques qui mesure l'erreur 'Err' de prédiction (MSE par exemple). Ensuite, le critère pour évaluer le compromis qualité prédictive/complexité est le suivant:

$$R_\alpha = Err + \alpha H$$

critère erreur prédictive avec:
 alpha = hyper-paramètre qui évalue la différence d'accuracy entre 2 sous-arbres imbriqués
 H = nb de feuilles

APPENDICE 1: ARBRES DE DÉCISIONS

Amélioration des performances du modèle avec des méthodes d'ensembles:

1. L'**ensachage** (ou **bagging**) : cette méthode consiste à construire plusieurs arbres de décision en utilisant des sous-ensembles aléatoires de l'ensemble de données d'apprentissage. Chaque arbre est construit en utilisant un échantillon différent, puis une procédure de consensus est utilisée pour combiner les prédictions de chaque arbre.
2. Les **forêts d'arbres aléatoires** (**random forests**) : cette méthode est une extension de l'ensachage. Elle utilise plusieurs arbres de décision construits à partir d'échantillons aléatoires de données et de variables d'entrée aléatoires. Les prédictions sont combinées en utilisant une méthode de majorité pour fournir une prédition finale.
3. Le **boosting** d'arbre de classification et de régression consiste à construire des arbres de décision successifs en se concentrant sur les erreurs de prédition des arbres précédents. Les prédictions des arbres sont pondérées en fonction de leur précision, ce qui donne plus de poids aux arbres qui ont une meilleure performance.
4. La **classification par rotation de forêts d'arbres de décision** utilise une analyse en composantes principales (PCA) sur un ensemble aléatoire de variables d'entrée pour réduire la dimensionnalité des données. Elle construit ensuite des arbres de décision sur les composantes principales pour améliorer la performance de la classification.

APPENDICE 2: FORÊTS D'ARBRES DÉCISIONNELS

Méthodes d'apprentissage ensembliste :

Une méthode d'apprentissage ensembliste est une technique qui combine plusieurs modèles d'apprentissage pour améliorer la précision des prédictions. L'idée principale derrière cette méthode est que la combinaison de plusieurs modèles peut donner de meilleurs résultats que chaque modèle pris individuellement (comme l'a dit un grand savant, 'seul on va plus vite, à plusieurs, on va plus loin').

Généralités sur le modèle :

L'objectif des forêts aléatoires est de rendre les arbres de décision plus indépendants les uns des autres pour améliorer la performance globale du modèle.

Pour cela, les forêts aléatoires introduisent deux éléments aléatoires lors de la construction de chaque arbre : la sélection aléatoire des **variables** et la sélection aléatoire des **observations d'entraînement**. Cela permet de construire des arbres qui se spécialisent sur différentes parties de l'espace de caractéristiques et réduisent ainsi la corrélation entre les arbres.

Les forêts aléatoires donnent de bons résultats, en particulier dans les problèmes de grande dimension où les arbres de décision CART peuvent avoir des performances médiocres.

Construction du modèle :

La méthode de construction d'une forêt aléatoire repose sur la sélection aléatoire d'échantillons avec remise dans la base d'apprentissage B, appelés également "**bootstrap samples**". Pour chaque échantillon bootstrap, on sélectionne au hasard q attributs parmi les p attributs existants de la base de données, et on construit un arbre de décision CART sur ces attributs.

Pour la régression, la prédiction finale est obtenue en agrégant les prédictions de chaque arbre de la forêt aléatoire par la moyenne, tandis que pour la classification, on utilise un vote majoritaire des prédictions de chaque arbre.

La sélection aléatoire des attributs pour chaque arbre permet de réduire la corrélation entre les arbres de la forêt, car chaque arbre sera construit sur un sous-ensemble différent des attributs.

En général, pour obtenir des résultats optimaux, la valeur de q est fixée à la racine carrée du nombre total d'attributs, soit $q=\sqrt{p}$ (pour de la **classification**) et à $q=p/3$ (pour de la **régression**).

Taille des arbres :

L'utilisation d'arbres peu profonds dans les forêts aléatoires permet de réduire la complexité de chaque arbre individuel, ce qui peut conduire à des performances moins élevées par rapport à des arbres plus profonds. Cependant, cette faiblesse est compensée par l'agrégation des prédictions de chaque arbre de la forêt, qui permet de combiner les forces de chaque arbre.

APPENDICE 2: FORÊTS D'ARBRES DÉCISIONNELS

Importance des attributs :

L'importance des attributs dans les forêts aléatoires est évaluée en utilisant des mesures telles que la **diminution de l'impureté de Gini** et l'**erreur OOB**. Ces mesures permettent de déterminer quels attributs sont les plus pertinents pour la prédiction de la variable cible, et peuvent être utilisées pour sélectionner les attributs les plus importants pour une tâche de prédiction donnée.

Pour chaque attribut, la somme des changements d'impureté de Gini est cumulée sur tous les arbres de la forêt aléatoire, ce qui donne une mesure globale de l'importance de l'attribut.

L'erreur OOB est une autre mesure d'importance des attributs dans les forêts aléatoires. Après la construction de chaque arbre, les échantillons qui n'ont pas été utilisés pour construire l'arbre (les échantillons OOB) sont utilisés pour évaluer la performance de l'arbre. Ensuite, les valeurs des attributs pour ces échantillons sont aléatoirement permutees, et la performance de l'arbre est de nouveau évaluée. La différence entre la performance initiale de l'arbre et sa performance après permutation des attributs est calculée pour chaque arbre et pour chaque attribut. La mesure globale de l'importance de l'attribut est la dégradation moyenne (changement du taux d'erreur) sur tous les arbres de la forêt aléatoire.

Sources :

- Arbres décisionnels

https://campus2.mines-ales.fr/pluginfile.php/75203/mod_resource/content/2/2IA%20-%20Data%20science%20-%20statistical%20learning.pdf (Cours de M Sutton-Charani)

<https://cedric.cnam.fr/vertigo/cours/ml2/coursArbresDecision.html>

<https://cedric.cnam.fr/vertigo/cours/ml2/docs/coursArbresDecision-1x2.pdf>

[https://fr.wikipedia.org/wiki/Arbre_de_d%C3%A9cision_\(apprentissage\)](https://fr.wikipedia.org/wiki/Arbre_de_d%C3%A9cision_(apprentissage))

- Forêts d'arbres décisionnels

https://fr.wikipedia.org/wiki/For%C3%AAt_d%27arbres_d%C3%A9cisionnels

<https://cedric.cnam.fr/vertigo/cours/ml2/docs/coursForetsAleatoires-1x2.pdf>