Introduction

nous essayons à travers de document d'expliquer la démarche que nous avons suivi pour résoudre de problème de nettoyage de données, le feature engineering, la standardisation de données, la modélisation, l'entraînement du modèle, l'optimisation des hyper-paramètres, l'application du modèle

Nettoyage de données

Notre étude est basée essentiellement sur les statistiques fournies sur les joueurs de Tennis qui se trouve principalement dans le fichier Stat.csv , Donc, dans une première étape nous avons commencé par le calcul des données manquantes (Missing Data) dans ce ficher puis nous avons éliminé les colonnes dont le pourcentage des données manquantes dépasse 90 % . (vous trouvez le code dans le fichier Features engineering).
Le reste des colonnes contient des données manquantes avec un pourcentage inférieur à 1 % , nous avons rempli ce manque avec « la stratégie de la moyenne ».
Après, nous avons cherché les joueurs dans la Dataframe d'entraînement (train.csv) dont les statistiques n'apparaissent pas dans le fichier Stat.csv (leurs IDs sont introuvables dans le fichier Stat.csv), après l'élimination des matchs qui contiennent ces joueurs nous avons réduit la dataframe d'entraînement (train.csv) de 3 %. Par contre, pour la dataframe de test (test.csv) les matchs qui contiennent des joueurs introuvables dans le fichier stat.csv représentent mois de 1 %.

L'approche globale

Notre modèle va estimer la probabilité que le premier joueur gagne sachant des informations relatives aux deux joueurs et au match.
Pour la plupart des variables, on va calculer un score pour le premier joueur et un second pour le deuxième puis on construit la variable par une simple soustraction entre les deux scores.
Par exemple, nous pensons à construire une variable relative au nombre des matchs gagnés parmi les matchs joués par un joueur, on calcule ce score (les matchs gagnés sur les matchs joués) pour le premier joueur puis pour le deuxième et enfin on soustrait les deux scores.

$$Match\ gagnés = \frac{nombre\ des\ matchs\ gagnés\ par\ Joueur\ 1}{nombre\ des\ matchs\ joués\ par\ Joueur\ 1} \_ \frac{nombre\ des\ matchs\ gagnés\ par\ Joueur\ 2}{nombre\ des\ matchs\ joués\ par\ Joueur\ 2}$$

Nous pouvons générer des informations (variables) par deux méthodes selon les données disponibles :

- extraction de la variable directement à partir des données : exemple la différence du rang entre les joueurs, on extrait cette variable directement à partir du fichier « Player rates.csv »
- création de la variable après un traitement des données : on estime la variable par une opération par exemple la moyenne comme l'exemple du paragraphe précédente nous avons calculé la variable « matches gagnés » par la soustraction de la moyenne des matchs gagnés par le premier joueur et celle du deuxième joueur.

Les variables générées :

- DF_ACE : la différence d'ace par jeux (nombre des matchs joués) entre les deux joueurs.
  - $$DF-ACE = \frac{\sum ace\ gagnés\ par\ Joueur\ 1}{nombre\ des\ matchs\ joués\ par\ Joueur\ 1} - \frac{\sum ace\ gagnés\ par\ Joueur\ 2}{nombre\ des\ matchs\ joués\ par\ Joueur\ 2}$$

Cette variable nous informe sur la capacité en attaque des joueurs.

- averge_DF : la différence des doubles fautes par jeux entre les deux joueurs.

$$averge - DF = \frac{\sum double\,Fautes\,réalisées\,par\,Joueur\,1}{nombre\,des\,matchs\,joués\,par\,Joueur\,1} - \frac{\sum double\,Fautes\,réalisées\,par\,Joueur\,2}{nombre\,des\,matchs\,joués\,par\,Joueur\,2}$$

- PFS : différence entre la somme le Nombre de points gagnés sur premier services sur le nombre de points joués sur premier service sur le nombre des matchs joués.

$$DF - PFS = \frac{\sum_i \frac{N°\,de\,points\,gagné\,par\,J\,1\,sur\,S\,1\,durant\,M\,i}{N°\,de\,points\,joués\,par\,J\,1\,sur\,S\,1\,durant\,M\,i}}{N°\,des\,matchs\,joués\,par\,Joueur\,1} - \frac{\sum_i \frac{N°\,de\,points\,gagné\,par\,J\,2\,sur\,S\,1\,durant\,Mi}{N°\,de\,points\,joués\,par\,J\,2\,sur\,S\,1\,durant\,Mi}}{N°\,es\,matchs\,joués\,par\,Joueur\,2}$$

M : match
S : service
j : joueur
i : parcourt les matchs joués par J1 puis par j 2
N° : nombre

- PSS : différence entre les deux joueurs de la division de la somme « ‘Nombre de points gagnés sur premier service’ sur ‘Nombre de points joués sur premier service’ » sur nombre des matchs joués.

$$DF - PSS = \frac{\sum_i \frac{N°\,de\,points\,gagné\,par\,J\,1\,sur\,S\,2\,durant\,M\,i}{N°\,de\,points\,joués\,par\,J\,1\,sur\,S\,2\,durant\,M\,i}}{N°\,des\,matchs\,joués\,par\,Joueur\,1} - \frac{\sum_i \frac{N°\,de\,points\,gagné\,par\,J\,2\,sur\,S\,2\,durant\,Mi}{N°\,de\,points\,joués\,par\,J\,2\,sur\,S\,2\,durant\,Mi}}{N°\,des\,matchs\,joués\,par\,Joueur\,2}$$

- Win : différence entre nombre des matchs gagnés parmi les matchs joués.

$$Win = \frac{nombre\,des\,matchs\,gagnés\,par\,Joueur\,1}{nombre\,des\,matchs\,joués\,par\,Joueur\,1} - \frac{nombre\,des\,matchs\,gagnés\,par\,Joueur\,2}{nombre\,des\,matchs\,joués\,par\,Joueur\,2}$$

- Break : différence entre les breaks gagnés parmi les breaks joués :

$$DF - PSS = \frac{\sum_i \frac{N°\,de\,balles\,de\,break\,gagné\,par\,J\,1\,durant\,M\,i}{N°\,de\,balles\,de\,break\,obtenues\,par\,J\,1\,durant\,M\,i}}{N°\,des\,matchs\,joués\,par\,Joueur\,1} - \frac{\sum_i \frac{N°\,de\,balles\,de\,break\,gagné\,par\,J\,2\,durant\,M\,i}{N°\,de\,balles\,de\,break\,obtenues\,par\,J\,2\,durant\,M\,i}}{N°\,des\,matchs\,joués\,par\,Joueur\,2}$$

- Head to Head : pour estimer la probabilité qu’un joueur J1 gagne (J2 perd), on fait appel à l’historique des matchs entre les deux joueurs, on calcule le nombre des matchs durant lesquels le joueurs J1 gagne moins le nombre des matchs gagnés par son adversaire (J2) et on les divise par le total des matchs entre les deux joueurs .

- La fatigue : certainement l'état physique des joueurs impacte le résultat du match, on peut modéliser la fatigue d'un joueur par le nombre des matchs joués par ce dernier durant les trois derniers jours avant sa confrontation avec l'adversaire en question. Pour être plus précis, on peut attribuer des différents poids aux jours (par exemple 3/6 pour Jours-1, 2/6 pour jours-2 et 1/6 pour jours-3).

- Différence des points : cette variable calculée directement à partir du fichier « player rates.csv ».

- Différence des rangs : cette variable calculée directement à partir du fichier « player rates.csv ».

- Complet : cette variable décrit « la complétude » d'un joueur , on dit qu'un joueur J1 est « complet » s'il est performant en attaque et en défense , la variable « complet » est la résultante du produit d'une première variable qui décrit l'attaque (Ace par match) une deuxième qui décrit la défense (break gagnés par joués par matchs)

- Différence entre nombre de points gagnés sur nombre de points joués

Calcul des variables :

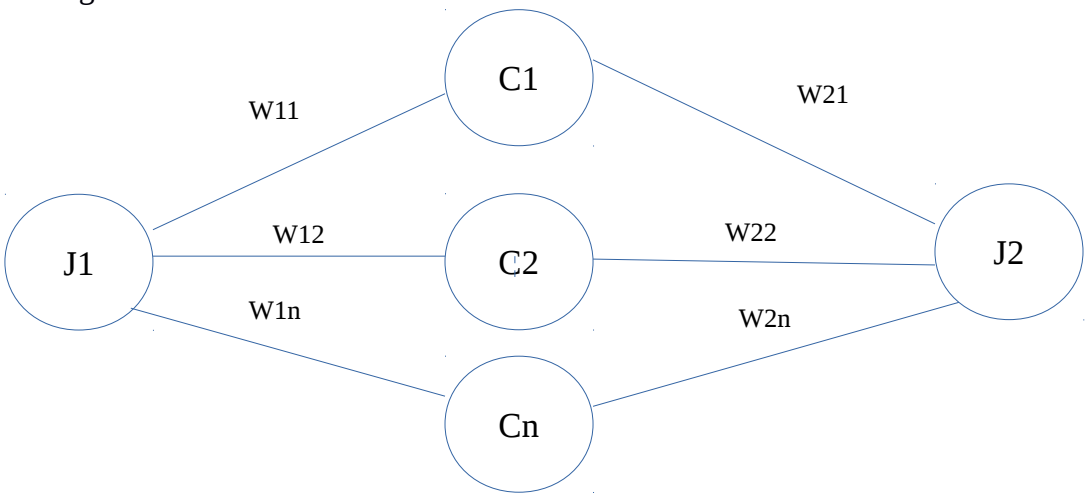
Pour calculer toutes ces variables, nous avons codé une fonction Python qui prend l'identifiant du premier et du deuxième joueur comme paramètre avec les dataframes à partir desquelles on va dégager les données.
Puis, nous avons calculé pour chaque couple des joueurs la variable en considération et nous avons la stockée dans une colonne de dataframe d'entraînement.

Résultat :

Après le calcul des variables, nous sommes arrivé à une dataset qui contient les IDs de joueurs gagnants, les IDs de joueurs perdus, les IDs de tournois et les variables construites. Finalement, pour sauvegarder le travail, nous avons exporté cette dataset vers un fichier CSV.

Autres variables peuvent être calculées :

- **<u>Comportement vis-à-vis les adversaires communs</u>** : l'approche des adversaires communs est très importante pour avoir une idée sur la performance des deux joueurs. Pour chaque couple des joueurs, on extrait la liste des adversaires communs. Pour chaque élément de cette liste, on fait le calcul du pourcentage des matchs gagnés par joueur 1 et 2. Comme tout autre variable, on fait la différence entre ces deux pourcentages.



W_{ij} = nombre des matchs gagnés par le joueurs J_i contre l'adversaire commun C_j sur le nombre totale des matchs joués entre les deux

$$. \quad CAC = \frac{1}{n}\left(\sum_{i=1}^{n}(W1i - W2i)\right)$$

Malheureusement, le calcul de cette variable prend énormément du temps pour cela nous avons évité de calculer cette variable.

- **<u>Différence d'âge entre les joueurs</u>** : Certainement la différence d'âge impacte le rendement du joueur. Selon les statistiques de Tennis, le joueur âgé entre 25 et 30 sont les plus performants. Contrairement aux autres variables, la simple soustraction ne donne pas un résultat pertinent car 7 ans de différence entre J1 d'âge 32 et J2 de 25 n'est pas la mémé pour J1 de 25 et J2 de 18 . Dans le premier cas J2 est plus performant par contre dans le deuxième cas J1 est plus performant. Pour résoudre ce problème on peut diviser les ages des joueurs en trois ou quatre tranches et donner le poids le plus important pour la tranche d'âge entre 22 et 27 par exemple.

Malheureusement, à cause d'absence des données (plus de 50% des dates de naissance de joueurs ne sont pas disponibles) nous ne pouvons pas calculer cette variable.

- Différence entre nombre de points gagnés sur nombre de points joués : de même on ne peut pas calculer cette variable à cause d'absence des données

standardisation des données :

Après la construction des variables, et avant d'implémenter n'importe quel modèle (sauf ceux basés sur l'arbre de décision) nous avons besoin de normaliser ou standardiser les données (plusieurs techniques peuvent être utilisées comme le Max-Min transformation ou la standardisation de variable c'est-à-dire les amener à suivre la loi normale)
la standardisation et la normalisation sont nécessaires pour minimiser « l'effet du poids » lors du calcul de distances, rapprocher les valeurs de variables les uns des autres (les mettre tous dans l'intervalle [0,1] par la transformation Max-Min par exemple) afin d'éviter la domination d'une variable sur les autres, dans notre cas la variable « différence de point » est l'ordre de 700 entre certains joueurs donc elle domine les autres variable en effet la distance euclidienne entre ces deux joueurs est proche de la différence des points entre eux, cela cause problème lors l'optimisation du modèle.
Dans le cas de cette compétition, nous avons choisi d'utiliser une transformation robuste aux outliers disponible sur la bibliothèque Scikit learn

NB : quelque soit le modèle de classification nous avons besoin pour l'implémenter les besoins des informations sur au moins deux classes par contre la méthode avec la quelle nous avons construit tout nos variables (soustraction) le premier joueurs est toujours le joueur gagnant, pour résoudre ce problème nous avons permuté, pour certain match , les joueurs de façon que le premier joueur devient le joueur perdu et nous avons multiplié les variables correspondant à cette ligne (ce match) par (-1) .
Pour implémenter cette technique nous générons une liste aléatoire des indices entre 0 et le nombre de matchs par l'intermédiaire de la bibliothèque de Numpy et nous avons inverser le joueurs des matchs dont l'indice se trouve dans cette liste.

Les modèles

plusieurs modèles peuvent être implémentés pour résoudre ce problème, dans une première approche nous avons commencé par « les modèles les plus simples » comme le modèle SVM (support vector machine) puis la régression logistique.

- SVM : à travers ce modèle on cherche l'hyperplan (dim = n-1) qui sépare mieux nos deux classes.
  Pour implémenter ce modèle , nous avons utilisé la bibliothèque « Sklearn »
  pour tester la qualité du modèle, nous faisons appel à la matrice de la confusion. Après la prédiction des classes de X_test on trouve de 7O % sont bien classés par le modèle.

- Régression Logistique : ce modèle estime la probabilité que le premier joueur gagne à travers la fameuse fonction logistique :

$$h_{\Theta}=\frac{1}{1+e^{-\Theta^{T}\cdot X}}=Pr\left(Y=1 \vee \left(X;\Theta\right)\right)$$

θ : vecteur à déterminer en minimisant une certain « coût » à travers un certain optimiseur
X : vecteur variable du match
Y : variable booléenne 1 si le premier joueur gagne 0 sinon

A fin d'implémenter le modèle de régression nous avons commencé par une première méthode : l'utilisation de la bibliothèque « Sklearn ». Pour tester le modèle nous faisons appel à la matrice de confusion , nous trouvons un score un peu mieux que celui de SVM.
Puis nous avons implémenté le modèle à travers TensorFlow (afin de contrôler mieux les hyper-paramètres)
L'optimiser utiliser c'est AdamOptimizer grâce à sa rapidité avec un learning-rate de 0.1.

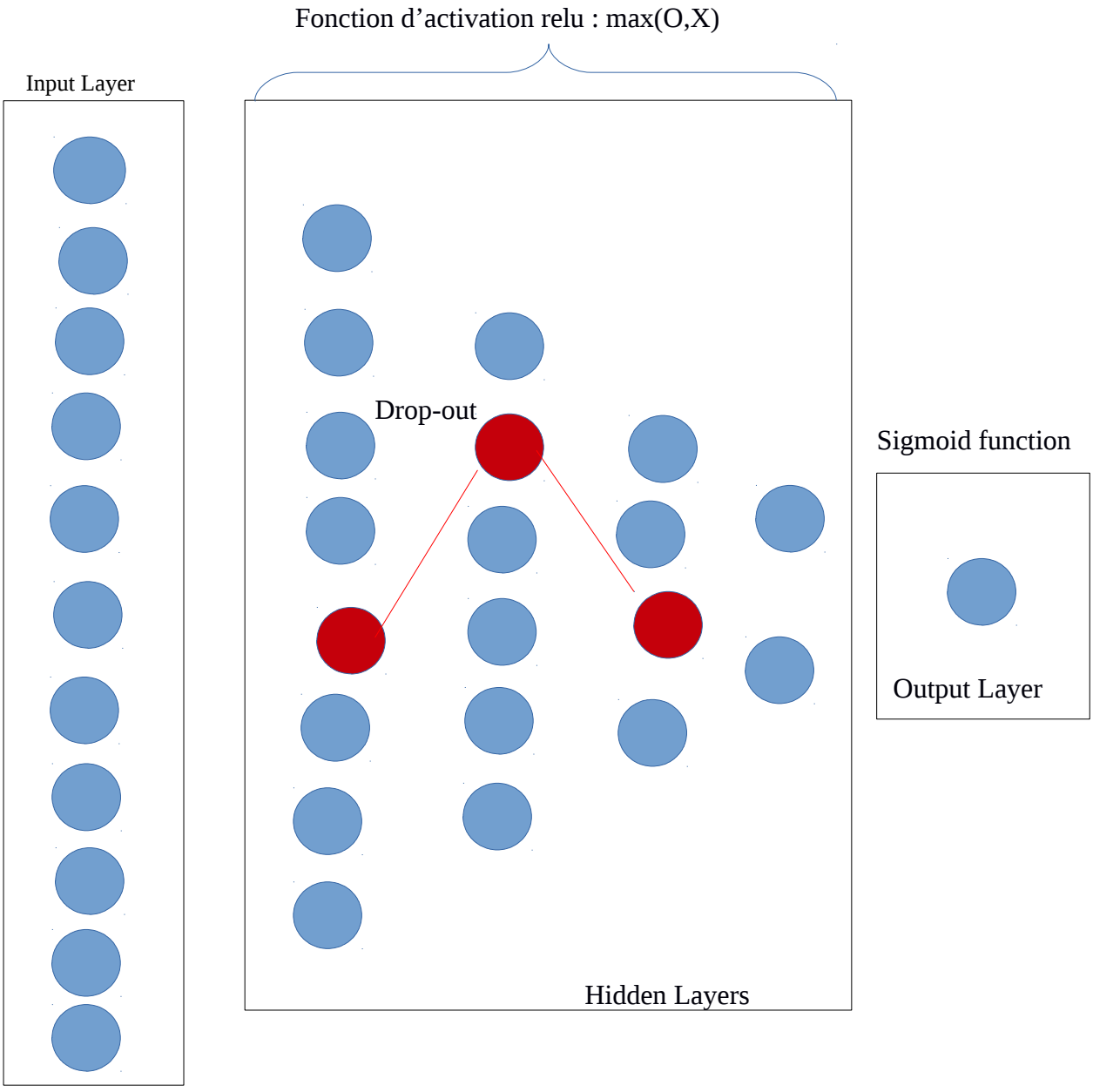
La fonction coût : cost(x,y)= $\frac{-1}{m}\cdot\sum_{i=1}^{m} yi\ln\left(h\Theta\left(x\right)\right)+\left(1-yi\right)\left(1-h\Theta\left(x\right)\right)$

les hyper-paramètres du ce modèle sont :

- Le learning _rate : la pas avec laquelle l'algorithme d'optimisation va change « descendre » cherchant le minimum.
- L'algorithme d'optimisation
- batch size : « la quantité » des données qu'on va « injecter » à chaque itération
- nombre d'itération

- réseau de neurones

  structure de réseau



les hyper-paramètres de ce modèle sont :

- l'optimiseur : l'algorithme d'optimisation il est possible d'utiliser Adam ou Rmsprop
- batch size
- drop_rate : Le pourcentage avec lequel on va déactiver certains neurones dans les couches intermédiaires (Hidden Layers) , protège le réseau de neurones contre le sur-apprentissage (overfitting)
- nombre de epochs.
- l'initialisation : la méthode avec laquelle on va initialiser nos poids
- nombre des couches intermédiaires cachées : Hidden Layers

- les fonctions d’activation

Implémentation du modèle :

Pour implémenter le modèle, nous avons choisi la bibliothèque Keras avec TensorFlow backend, afin de minimiser le code et se concentrer sur la modélisation. Parmi ces hyper-paramètres nous avons fixé le nombre des couches intermédiaires et les fonctions d’activation pour le temps d’optimisation des hyper-paramètres.
Afin d’optimiser les hyper-paramètres nous avons utilisé GridSearch de la bibliothèque de Sklearn en essayant toutes combinaisons possibles , après un temps d’exécution qui dépasse **48 heures !!!**
nous sommes arrivés aux hyper-paramètres suivants :

- batch size : 5
- nombre d’epochs : 120
- optimiseur : Rmsprop
- drop_rate : 0.1
- initialisation : glorot uniform

le résultat est logique :

- Rmsprop est comme sa performance dans les problèmes de classification (il utilise des différents learning_rate)
- drop_rate = 0.1 nous avons désactivé aléatoire 10 % des nœuds donc nous sommes protégé contre l’overfitting
- initialisation d’Xavier : on initialise les poids tels que $Var(W_i)=\frac{1}{N}$ avec N est la moyenne entre nombre d’inputs et d’output de la couche, ce résultat est logique puisque la distribution des données à l’entrée est normale de moyenne proche de zéro

nous avons appliqué ce modèle à la dataframe de test et nous avons exporté le résultat dans fichier csv.