# Projet de programmation: Prédiction de l'issue de matchs de tennis

Victor Michel et Justin Morelon

**ARTICLE HISTORY**
Compiled May 25, 2021

## 1. Introduction

Qui va gagner Roland Garros cette année, et pourquoi Nadal ? C'est de cette question que notre projet est né. Le tennis est un sport où les statistiques sont omniprésentes, que ce soit sur la vitesse des services, le pourcentage de coups gagnants de Federer contre Djokovic,etc... Dans ce sport, les faibles écarts de niveaux se reflètent très vite au score: en moyenne Federer a gagné 54% de ses échanges en carrière. Le tennis nous semblait donc tout particulièrement porté a être modélisé. Nous avons donc décidé d'essayer de prédire l'issue des matchs de tennis à partir des données préexistantes et de calculer le gain si nous avions parié sur les matchs modélisés.

## 2. Déroulement du projet

Quelques erreurs nous ont fait perdre du temps. Par exemple, le fait d'initialiser un array avec des entiers du type integer empêche l'attribution ultérieure de flottants. Cela a faussé les calculs de moyenne de toute une partie de l'algorithme. De même, une autre erreur qui a été difficile a détecter est la modification de liste de liste. Il est alors judicieux d'utiliser la deepcopy de la liste pour pouvoir la modifier comme souhaité.

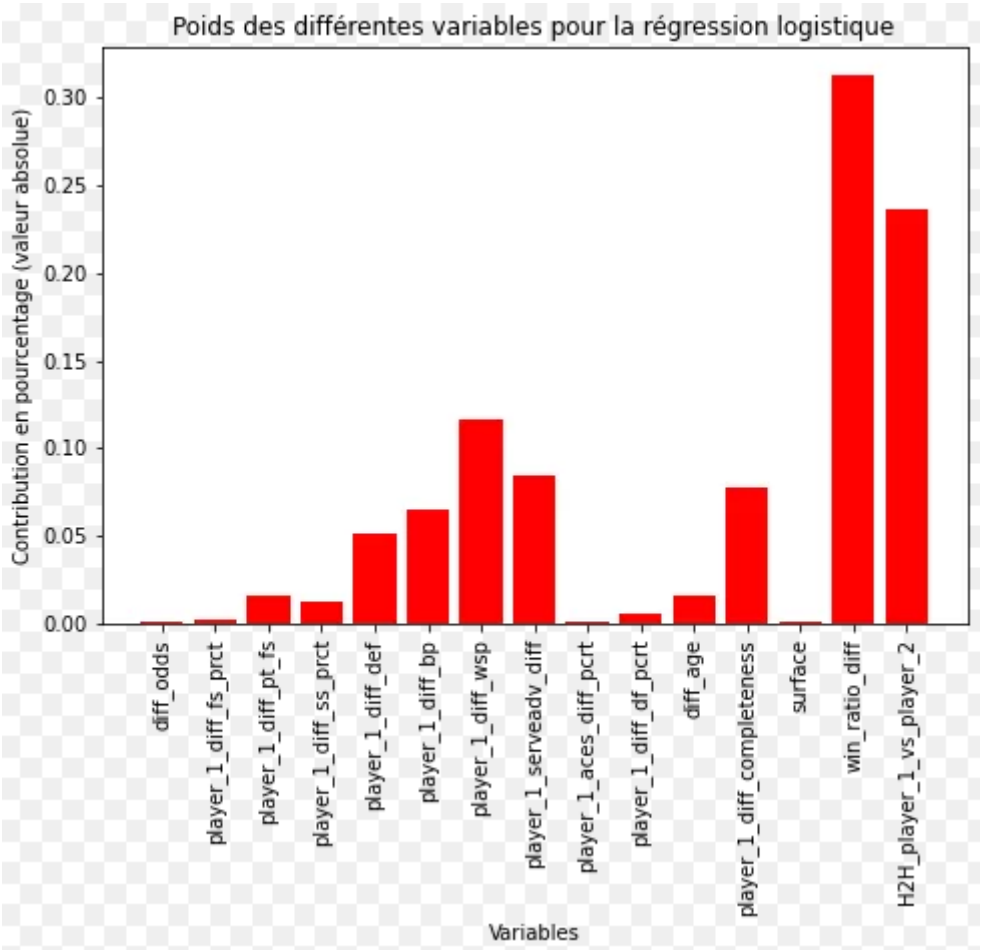
Certaines parties du code ont dû être réécrites car elles n'étaient pas assez efficaces. En cause, des boucles for parfois inutiles. Lorsqu'on manipule beaucoup de données, la complexité inutilement grande des algorithmes que nous avions codés rendaient en pratique impossible l'obtention de résultats dans un délai raisonnable. La taille importante de la base de données complète ( 110,000 matchs) nous a forcé à écrire nos alogrithmes de la façon la plus optimisée possible.

## 3. Données, modélisation et hyperparamètres

Afin de prédire l'issue des matchs de tennis, nous avions besoin d'un nombre important de données des matchs précédemment joués pour entraîner notre algorithme, que nous avons obtenu d'après la base de données du logiciel Oncourt. La base de données est divisée en trois parties: train (pour entraîner nos algorithmes de prédiction), valid (pour optimiser les hyperparamètres), et enfin test pour tester nos algorithmes et éviter l'overfitting. Nous avons choisi de diviser nos bases de données selon une répartition

65%-30%-5%. Nous avons d'abord utilisé R, puis Python ainsi que les modules panda et sklearn afin de mener à bien notre projet. D'abord R, afin de nettoyer la base de données des valeurs aberrantes, de fusionner les différentes bases de données, puis Python et ses modules sklearn et pandas pour construire les dernières variables, et d'entraîner et de tester nos algorithmes de classification.

Après avoir nettoyé la base de données des matchs incomplets, et en sélectionnant les matchs qui nous intéressait (nous avons par exemple supprimé les doubles et les matchs de Coupe Davis), nous avons décidé de retenir une série de variables pertinentes dont le poids lié à chacun des paramètres est attribué lors de l'entraînement de l'algorithme grâce à une régression logistique et un SVM.




Dans notre base de données, chaque ligne contient les informations d'un match, J1 contre J2. Chaque ligne contient des informations générales: le nom des joueurs, leur date de naissance, le nom et la date du tournoi. Elle contient également les variables de matchs, qui sont toutes exprimées sous forme de différence: par exemple, la variable diff_age est l'age de J1 moins l'age de J2. La base de données comporte également une variable 'player_1_win' qui vaut 1 ou 0, 1 si J1 a gagné, 0 sinon. Ainsi nous demandons à notre algorithme de calculer les poids qui permettent de maximiser la valeur de la

sigmoïde sachant que J1 a gagné, ce qui est un problème plus facile à résoudre.

Après avoir lu les différents articles de recherche existant déjà sur le sujet (principalement celui de Michal Sipko, Machine Learning for Prediction of Professional Tennis), nous avons introduit deux variables supplémentaires par rapport à ce qui avait déjà été produit par le passé. Tout d'abord nous avons voulu prendre les cotes du site Pinnacle en compte. En effet, les cotes des sites de paris sportifs nous donnent des informations sur la probabilité que le joueur X gagne contre le joueur Y. Outre le fait que les cotes nous donnent des probabilités implicite que tel joueur gagne face à telle autre (l'inverse de la cote du joueur X est une approximation correcte de sa probabilité de gagner), la cote nous donne des informations sur l'état de forme récente du joueur, de sa forme physique, etc. . . , des informations récentes que notre algorithme common opponents ne nous fournit pas, puisqu'il s'appuie sur les matchs passés qui peuvent dater de quelques années. Seulement, si cette idée paraissait bonne, le poids associé par la régression logistique à la variable diff_odds est quasi nulle. Nous avons également fait le choix de ne pas retenir le rang ATP des joueurs comme variable, car le poids qu'elle occupait sinon était beaucoup trop important, pour une accuracy d'environ 0.6.

La seconde variable implantée s'est avérée beaucoup plus efficace. Diff_wsp mesure la différence de la probabilité de gagner sur son service entre un joueur X et un joueur Y. Cette variable représente plus de 10% du total des poids de la régression logistique. En outre, la construction d'une variable représentant la probabilité de gagner sur son service a été nécessaire: $\mathrm{WSP}_i$.

$$WSP_i = W1SP_i.FS_i + W2SP_i.(1 - FS_i)$$

où W1SP représente le pourcentage de premiers services gagnés lorsqu'ils sont passés et où FS représente le pourcentage de chance de passer son premier service plutôt que le second. Dans la conception du code, nous avons choisi de prendre en compte différemment les matchs sur une autre surface que celle où nous choisissons de faire la prédiction, Les matchs sur la même surface sont comptés avec un poids de 1 tandis que ceux sur une autre surface sont comptés avec un poids bêta plus petit que 1. De même les anciens matchs devaient jouer un poids moins important, on a donc décidé d'utiliser un coefficient en exponentielle de moins alpha fois le temps, alpha étant un coefficient positif. Alpha et bêta sont nos deux hyperparamètres, et après optimisation, valant respectivement 0.22 et 0.5 .

## 4. Description de l'algorithme

L'algorithme fonctionne en deux parties. La première consiste, à partir de la donnée du nom de deux joueurs, d'une surface ainsi que d'une date, à prédire les statistiques probables du match en moyennant avec pondération les statistiques des joueurs face à leurs précédents adversaires ainsi que leurs face à face. Pour ce faire, nous utilisons la méthode des adversaires communs (common opponents en anglais) . En effet, cette approche nous a paru plus pertinente que de simplement calculer la moyenne pondérée des joueurs face à tous leurs adversaires durant leurs carrières. Une moyenne simple est en effet trop sensible à la différence entre les adversaires rencontrés. La méthode des adversaires communs consiste à lister les adversaires communs entre deux joueurs, et

de calculer la moyenne pondérée des performances des joueurs contre ces adversaires. De ce scénario concernant les statistiques du match, un second algorithme peut alors prédire une estimation du pourcentage de chance de victoire des joueurs.

Les 3 fichiers partition, standardisation et H2H ne servent qu'à générer les bases de données sur lesquelles nous avons ensuite travaillé. Les principales bases de données sont all_data_ml, ainsi que all_data_co (utilisé pour notre algorithme commonopponents). La génération des bases de données prend du temps, environ 5 heures, du fait de la construction de nouvelles variables comme win_ratio_diff ou H2H ( voir codebook pour l'explication détaillée des variables). Ce fut une des principales difficultés rencontrées lors du codage. En effet, nos algorithmes de construction n'était pas correctement optimisé, du moins au début. Ainsi, nous ne pouvions pas vérifier si nos algorithmes avaient marché correctement sur l'entièreté de la base de données. Nous avons finalement réécrit nos algorithmes et cela nous a fait gagner un temps considérable ( voir partie coût algorithmique).

Le fichier ML sert à entrainer les deux modèles que nous avons à notre disposition, le SVM et la régression logistique. Nous utilisons pour ce faire les modules de sklearn. Le fichier.py Functions_list contient la liste de toutes les fonctions utilisées dans nos algorithmes notamment les deux principales prediction et common_opponents. Enfin le fichier prediction sert à mesurer l'efficacité des modèles et de simuler l'évolution d'une 'bankroll' en fonction des prédictions de ces algorithmes. Imaginons maintenant que l'on veuille obtenir la probabilité que Rafael Nadal gagne un match sur terre battue face à Novak Djokovic, à la date du 2021-06 (le mois et l'année sont suffisants pour la date), et l'on sait que Nadal est coté à 1.2 sur Betclic et Federer à 3.50, également sur Betclic. Pour ce faire il faut importer la base de données data_age, ainsi qu'une base de donnée finissant par _co ( de préférence all_data_co, qui est plus complète). Il suffit ensuite d'entrer cette ligne de code :

```
Entrée [33]: prediction([('Rafael Nadal','Novak Djokovic','2021-06',1,1.2,3.5)],data_age,all_data)

C:\Users\Victor\anaconda3\lib\site-packages\pandas\core\frame.py:4125: SettingWithCopyW
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user
rsus-a-copy
  return super().rename(

Out[33]: ([(0.540580094861148, 'Rafael Nadal')],
 0    0.82919
 Name: uncertainty, dtype: object)
```

L'algorithme renvoie le nom du premier joueur ainsi que la probabilité qu'il gagne face à Djokovic. Il renvoie également une mesure de l'uncertainty qui est une variable mesurant la qualité des informations utilisés pour construire cette probabilité (ici 0.8). L'algorithme peut renvoyer le résultat de plusieurs matchs en même temps. Le modèle utilisé ici étant celui de la régression logistique.

## 5. Coût et design de H2H

Nous avons codé une première version de H2H en parcourant les matchs avec une boucle for la complexité de la première version de l'algorithme est en O(n), avec n le nombre total de matchs, le temps d'éxecution était insatisfaisant. Nous avons donc décidé de le coder d'une seconde manière en imposant seulement des conditions d'égalité sur le dataframe sans jamais utiliser de boucles. Le temps d'exécution a été environ réduit par 10 (le temps d'exécution dépend des joueurs donnés à l'algorithme).

```
79  def h2h(name1, name2,df):
80      V1=0
81      V2=0
82      for i in range(len(df['player_1_name'])):
83          if (df['player_1_name'].iloc[i]==name1 and df['player_2_name'].iloc[i]==name2):
84              if df['player_1_win'].iloc[i]==1:
85                  V1=V1+1
86              else:
87                  V2=V2+1
88          if (df['player_1_name'].iloc[i]==name2 and df['player_2_name'].iloc[i]==name1):
89              if df['player_1_win'].iloc[i]==1:
90                  V2=V2+1
91              else:
92                  V1=V1+1
93      print(V1)
94      print(V2)
95      if V1+V2>0:
96          return [V1+V2,(V1/(V1+V2))]
97      return False
98
```

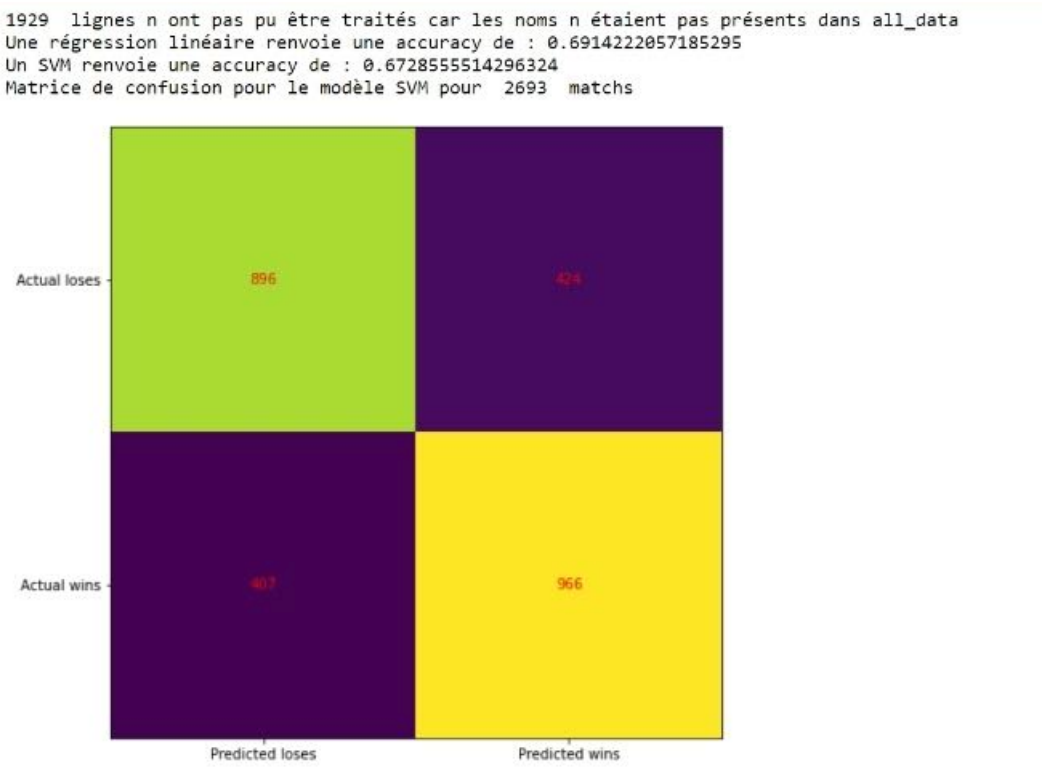
Le design de h2h est le suivant: pour chacun des matchs où les deux noms name1 et name2 apparaissent, V1 et V2 sont incrémentées de 1 suivant si le joueur avec le nom name1 gagne un face à face contre le joueur du nom de name2. Ensuite, en retournant V1+V2, l'algorithme donne accès au nombre total de match. Tandis que V1/(V1+V2) donne le pourcentage de victoire du joueur portant le name1.

## 6. Résultats des algorithmes

En entrainant nos algorithmes sur des matchs non simulés par notre algorithme common_opponents (comprendre sur des valeurs de vrais matchs), nous obtenons une très très bonne accuracy, sans overfiting, de 0.95. C'est assez peu étonnant, il est assez facile, en regardant simplement les statistiques d'un match déjà passé de savoir qui a gagné. En revanche, lorsque nos algorithmes de prédiction traitait des matchs simulés par common_opponents, nous perdons plus de 20 points d'accuracy. En testant nos aglorithmes sur la base de données data_test_co avec comme base de prédiction data_train_co, nous arrivons à traiter 2693 matchs sur 4622 matchs. Les matchs non traités correspondent aux matchs où l'un des deux joueurs n'est pas présent dans la base de données de prédiction.

Sur les matchs que nous avons pu prédire, nous obtenons une accuracy de 0.69 pour la régression logistique et 0.67 pour le SVM. Nous gardons par la suite le modèle de

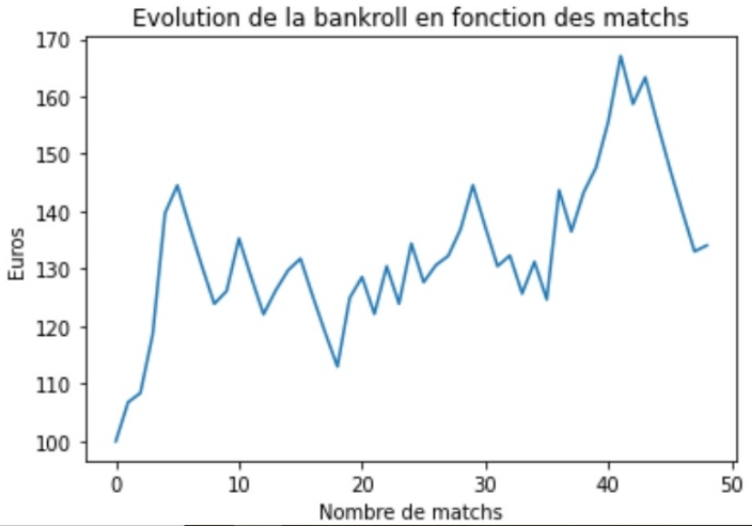
la régression logistique.




Nous avons ensuite simulé des paris sur ces 2693 matchs afin de vérifier si notre accuracy permettait de faire des bénéfices. Nous parions de la manière suivante: à chaque pari, nous ne misons qu'un pourcentage de notre fortune restante, si bien qu'il est théoriquement impossible de descendre en dessous de zéro (même s'il est tout à fait possible de perdre la mise de départ). De plus nous ne parions pas sur tous les matchs, mais seulement sur ceux dont nous estimons avoir assez d'informations, c'est-à-dire des matchs sur laquelle la variable uncertainty est inférieur à 1. Le graphique ci-dessous se base sur les prévisions de la régression logistique.

```
Quel est votre bankroll? 100
Quel est le pourcentage maximum que vous êtes pret à perdre? 0.05
```

Out[54]: `Text(0, 0.5, 'Euros')`




## 7. Prolongements et applications

Tout d'abord, il faut pousser l'optimisation des hyper paramètres. En effet, nous n'y avons pas passé de temps. Il est évident que les modèles utilisés ici ne sont pas assez complexes pour refléter la réalité complexe d'un match de tennis. Il aurait également été très intéressant de procéder à une ablation study, c'est-à-dire de regarder les combinaisons de variables que l'on peut retenir, et de recalculer l'accuracy, de les comparer, et enfin de garder le set de variables donnant le meilleur résultat. Il faudrait plutôt se tourner vers des modèles de Machine Learning comme des ANN, comme ce fut fait dans les travaux précédents. La base de données mériterait elle aussi d'être actualisée. Nous n'avons aucune statistique des matchs de 2021, ce qui handicape grandement notre prédiction pour les matchs actuels. Une bonne manière d'optimiser notre gain serait également d'améliorer la variable que nous appelons 'uncertainty'. En effet celle ci ne mesure que la qualité des matchs, et non pas la qualité de la probabilité rendue par nos algorithmes. Des variables supplémentaires pourrait être implémentées, comme la variable 'seed' qui mesure à quel point un joueur est favori avant d'entrer dans le tournoi. Enfin la base de données aurait pu être divisé de manière totalement différentes, que ce soit par tournoi, par saisons, ou même par couple de joueurs si nous ne sommes intéressés que par le résultat précis de quelques matchs.

A l'heure actuelle, notre modèle n'est applicable ni utilisable pour parier sur les matchs de tennis. Nous obtenons des résultats plutôt satisfaisant, mais il faut parier sur énormément de matchs pour pouvoir vraiment être rentable, et en simuler quasiment

6 fois plus (puisque nous ne parions pas sur tous les matchs), de plus le gain semble très volatile. Une manière assez conqéquente d'augmenter le gain même avec une faible accuracy est de descendre la valeur d'uncertainty pour définir sur quels matchs nous parions.

En conclusion, cette première approche pour la modélisation des matchs de tennis nous a motivé à améliorer nos résultats. Certains articles scientifiques mentionnent une accuracy de plus de 0.75, et nous sommes déterminés à atteindre cette valeur.

# *ANNEXE*

| | |
|---|---|
| Diff_odds | Différence de la cote du joueur 1 avec celle du joueur 2 (cotes venant du site anglais Pinnacle) |
| Player_1_diff_fs_prct | Différence du pourcentage de premier service réussi du joueur 1 avec celui du joueur 2 (S) |
| Player_1_diff_ss_prct | Différence du pourcentage de second service du joueur 1 avec celui du joueur 2 (S) |
| Player_1_diff_bp | Différence entre le pourcentage de balle de break réussi du joueur 1 et celui du joueur 2 (S,E) |
| Player_1_diff_wsp | Différence entre la probabilité de gagner sur son service du joueur 1 et de celle du joueur 2 (S, variable construite) |
| Player_1_serveadv_diff | Variable mesurant la force du service du joueur 1 et le retour du joueur 2 (variable construite) |
| Player_1_diff_aces_prct | Différence entre la probabilité du joueur 1 de faire un ace et celle du joueur 2 (S) |
| Player_1_diff_df_prct | Différence entre la probabilité du joueur 1 de faire une double-faute et celle du joueur 2. (S) |
| Diff_age | Différence d'âge entre le joueur 1 et le joueur 2 |
| Player_1_diff_completeness | Variable mesurant la 'complétude' du joueur 1 par rapport au joueur 2 (S, variable construite) |
| Surface | Variable discrète entre 1 et 5 indiquant la surface jouée. |
| Win_ratio_diff | Différence du pourcentage de victoire en carrière du joueur 1 face au joueur 2. |
| H2H | 'Head to Head' Différence entre le pourcentage de victoire du joueur 1 sur le joueur 2 et celle du joueur 2 sur le joueur 1. |

Toutes les variables dont la définition indique (S) sont standardisées, c'est-à-dire divisées par leur écart-type.

(E) signifie que nous avons nettoyé cette variable des valeurs aberrantes ou peut pertinentes. Par exemple, un joueur qui avait un taux de conversion de 100% de ses balles de breaks sur un match, mais qui n'avait eu qu'une seule balle de break, le match associé n'était pas comptabilisé.

La surface numéro 1 représente la terre battue, la numéro 2 le quick, la numéro 3 le indoor, le numéro 4 le gazon, et la numéro 5 une surface mixte entre le indoor et le quick.

Références :

Machine Learning for the Prediction of Professional Tennis Matches: MICHAL SIPKO

Game,Set,Learn: A machine learning approach to tennis match prediction: SHENOY,WANG,MALIK