Quand les mathématiques se mettent au service du tennis

Paul-Emile Marcus // Candidat n°31462

Figure 1: Résultat Open d'Australie 2019

Figure 2: Résultat Wimbledon 2019

Calcul du classement Elo

Probabilité de victoire de A sur B :

$$p(D) = \frac{1}{1 + 10^{-\frac{D}{400}}}$$

Le classement Elo est calculé à l'aide de la formule suivante :

$$E_{n+1} = E_n + K \times [W - p(D)]$$

$W \in \{0, 1\}$ représente l'issue du match

Force relative d'un joueur j sur i

Quelques complexités

- Meilleur classement en carrière : $\mathcal{O}(n)$
- Ratio de victoires au cours de la saison/en carrière : $\mathcal{O}(n)$
- Face-à-face entre deux joueurs donnés : $\mathcal{O}(n^2)$

Force relative d'un joueur j sur un joueur i

Prise en compte de plusieurs paramètres

$$\mathbb{P}_{\text{ELO}} = \frac{1}{1 + 10^{-\frac{D_{\text{ELO}}}{400}}}$$

$$\mathbb{P}_{\text{H2H}} = \frac{1}{1 + 10^{-\frac{D_{\text{H2H}}}{10}}}$$

$$\mathbb{P}_{\text{RATIO}} = \frac{1}{1 + 10^{-\frac{D_{\text{RATIO}}}{0.4}}}$$

$$\Rightarrow \mathbb{P}(j/i) = \frac{1}{x_{\text{ELO}} + x_{\text{H2H}} + x_{\text{RATIO}}} \cdot \sum_{\text{paramètres}} x_{\text{param}} \cdot \mathbb{P}_{\text{param}}$$

Force relative d'un joueur j sur un joueur i

Calcul

```
422   ### Cette fonction calucle la proba de victoire d'un joueur j sur un joueur i en se basant sur la formule du
      classement élo adaptée à plusieurs statistiques
423
424   def calcul_proba(player1, player2):
425     stats1, stats2 = stats_joueurs_pour_proba_final[player1], stats_joueurs_pour_proba_final[player2]
426
427     # Calcul proba classement elo
428     p1_elo, p2_elo = calcul_proba_elo(stats1[3], stats2[3], 400)
429
430     # Calcul proba ratio saison
431     p1_rs, p2_rs = calcul_proba_elo(stats1[5], stats2[5], 0.4)
432
433     # Calcul proba h2h
434     h2h_tot = stats1[7][stats2[0]] + stats2[7][stats1[0]]
435     if h2h_tot >= 5:
436       p1_h2h, p2_h2h = calcul_proba_elo(stats1[7][stats2[0]], stats2[7][stats1[0]], 10)
437
438       # Calcul proba finale
439       p = (50*p1_elo + 25*p1_rs + 25*p1_h2h)/100
440     else:
441       p = (70*p1_elo + 30*p1_rs)/100
442
443     return (p, 1-p)
444
```

Figure 3: Calcul de la force relative de j sur i

Matrice de transition

$$
\begin{array}{c}
 \\
\textit{Djokovic} \\
\textit{Federer} \\
\textit{Nadal} \\
 \\
 \\
 \\
 \\
\end{array}
\begin{array}{c}
\begin{array}{ccccccc}
\textit{Djokovic} & \textit{Federer} & \textit{Nadal} & & & & \\
\end{array} \\
\begin{pmatrix}
0 & 0.32 & 0.36 & \cdots & * & \cdots & * \\
0.68 & 0 & 0.57 & \cdots & * & \cdots & * \\
0.64 & 0.43 & 0 & \cdots & * & \cdots & * \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
* & * & * & \cdots & * & \cdots & * \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
* & * & * & \cdots & * & \cdots & 0
\end{pmatrix}
\end{array}
$$

Figure 4: Matrice de transition

Théorème de Perron-Frobenius

Théorème de Perron-Frobenius

Soient $A \in \mathscr{M}_n(\mathbb{R})$ positive et primitive et $r = \rho(A)$ son rayon spectral, alors :

- $r > 0$ est une valeur propre dominante et simple.
- $\exists ! \, x_0 > 0 \ / \ \|x_0\|_1 = 1$ et $Ax_0 = rx_0$.

Vers l'implémentation de l'algorithme

Théorème de Perron-Frobenius

Soit $Y_0 \in \mathscr{M}_{n,1}(\mathbb{R}^+)$, tel que $\|Y_0\|_1 = 1$. Alors, $A^n Y_0 \xrightarrow[n\to+\infty]{} Y_\infty$, avec $Y_\infty \in \mathscr{M}_{n,1}(\mathbb{R}_+^*)$, $AY_\infty = rY_\infty$, et $\|Y_\infty\|_1 = 1$.

L'algorithme de Perron-Frobenius

```
473
474     # Initialisation du vecteur
475     Y = np.array([[1/n] for _ in range(n)])
476
477     # Itération jusqu'à convergence
478     for i in range(1000):
479         tmp = np.dot(matrice_transition, Y)
480         Y = tmp / np.linalg.norm(tmp)
481         if i == 9 or i==99 or i==999:
482             print ([Y[0],Y[1],Y[2]])
483
484     Y_normalized = Y / np.sum(Y)
485
486     # On trie les indices des états en fonction des valeurs du vecteur propre
487     classement_indices = np.argsort(Y_normalized, axis=0)[::-1]
488
489     # Création du classement
490     classement = [(indice, Y_normalized[indice][0]) for indice in classement_indices]
491
492     C = []
493     for i in range(len(classement)):
494         C.append(dict_index_nom[classement[i][0][0]])
495
496     return(C)
497
```

Figure 5: Algorithme de Perron Frobenius

Algorithme de Perron Frobenius

Quelques itérations

$$AY_0 = \begin{pmatrix} 0.0235 \\ 0.0252 \\ 0.0258 \\ \vdots \\ * \end{pmatrix} \qquad A^{10}Y_0 = \begin{pmatrix} 0.0246 \\ 0.0250 \\ 0.0260 \\ \vdots \\ * \end{pmatrix} \qquad A^{100}Y_0 = \begin{pmatrix} 0.0246 \\ 0.0250 \\ 0.0260 \\ \vdots \\ * \end{pmatrix}$$

Les classements du 01/10/23

Table 1: Mon classement

1	Djokovic N.
2	Alcaraz C.
3	Medvedev D.
4	Sinner J
5	Zverev A.
6	Rublev A.
7	Hurkacz H.
8	Dimitrov G.
9	Fritz T.
10	Rune H.

Table 2: Classement ATP

1	Djokovic N.
2	Alcaraz C.
3	Medvedev D.
4	Sinner J
5	Rublev A.
6	Tsitsipas S.
7	Rune H.
8	Ruud C.
9	Zverev A.
10	Fritz T.

Variations de classement

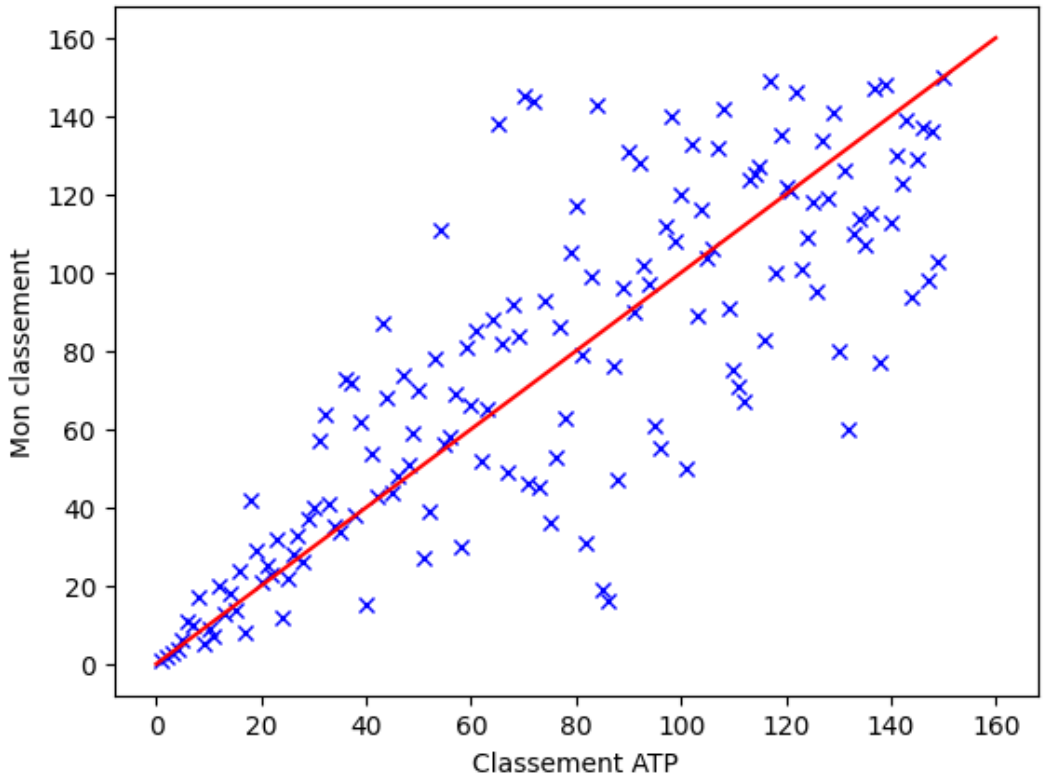

Figure 6: Nouveau classement en fonction du classement ATP

Écart de classement

- Moyenne : 7,79 et Écart-type : 32,4

Prédiction d'un match

Un problème binaire

Classement J1 (ATP)	Classement J2 (ATP)	1er tour	2eme tour	...
14	27	1	0	

Si le mieux classé gagne

Vainqueur
1

Si le mieux classé perd

Vainqueur
0

Distance de Levenshtein

```python
def distance_levenshtein(mot1, mot2):
    # On initialise une matrice de taille (longueur_mot1 + 1) x (longueur_mot2 + 1)
    dist = [[0] * (len(mot2) + 1) for _ in range(len(mot1) + 1)]

    # On initialise la première colonne avec les distances de 0 à la longueur du mot1
    for i in range(len(mot1) + 1):
        dist[i][0] = i

    # On initialise la première ligne avec les distances de 0 à la longueur du mot2
    for j in range(len(mot2) + 1):
        dist[0][j] = j

    # On calcule la distance de Levenshtein
    for i in range(1, len(mot1) + 1):
        for j in range(1, len(mot2) + 1):
            cout_substitution = 0 if mot1[i - 1] == mot2[j - 1] else 1
            dist[i][j] = min(dist[i - 1][j] + 1,            # effacement du nouveau caractère de mot1
             dist[i][j - 1] + 1,                            # insertion dans mot2 du nouveau caractère de mot1
             dist[i - 1][j - 1] + cout_substitution)        #substitution

    # La distance de Levenshtein est le dernier élément de la matrice
    return dist[len(mot1)][len(mot2)]

def trouver_correspondance_nom(nom_inexact, liste_noms):
    correspondance_score_min = float('inf')
    nom_correspondant = None
    for nom_exact in liste_noms:
        distance = distance_levenshtein(nom_inexact, nom_exact)
        if distance < correspondance_score_min:
            correspondance_score_min = distance
            nom_correspondant = nom_exact
    return nom_correspondant
```

Figure 7: Implémentation de la distance de Levenshtein

Random Forest

Localité	Superficie	Chambres	Jardin	Parking	Salles de bain
Le Mans	70	2	non	non	1
Sigean	128	4	oui	oui	2
Bayonne	256	5	oui	oui	3
Saumur	28	1	non	oui	1

Figure 8: Division aléatoire du jeu de données

Random Forest

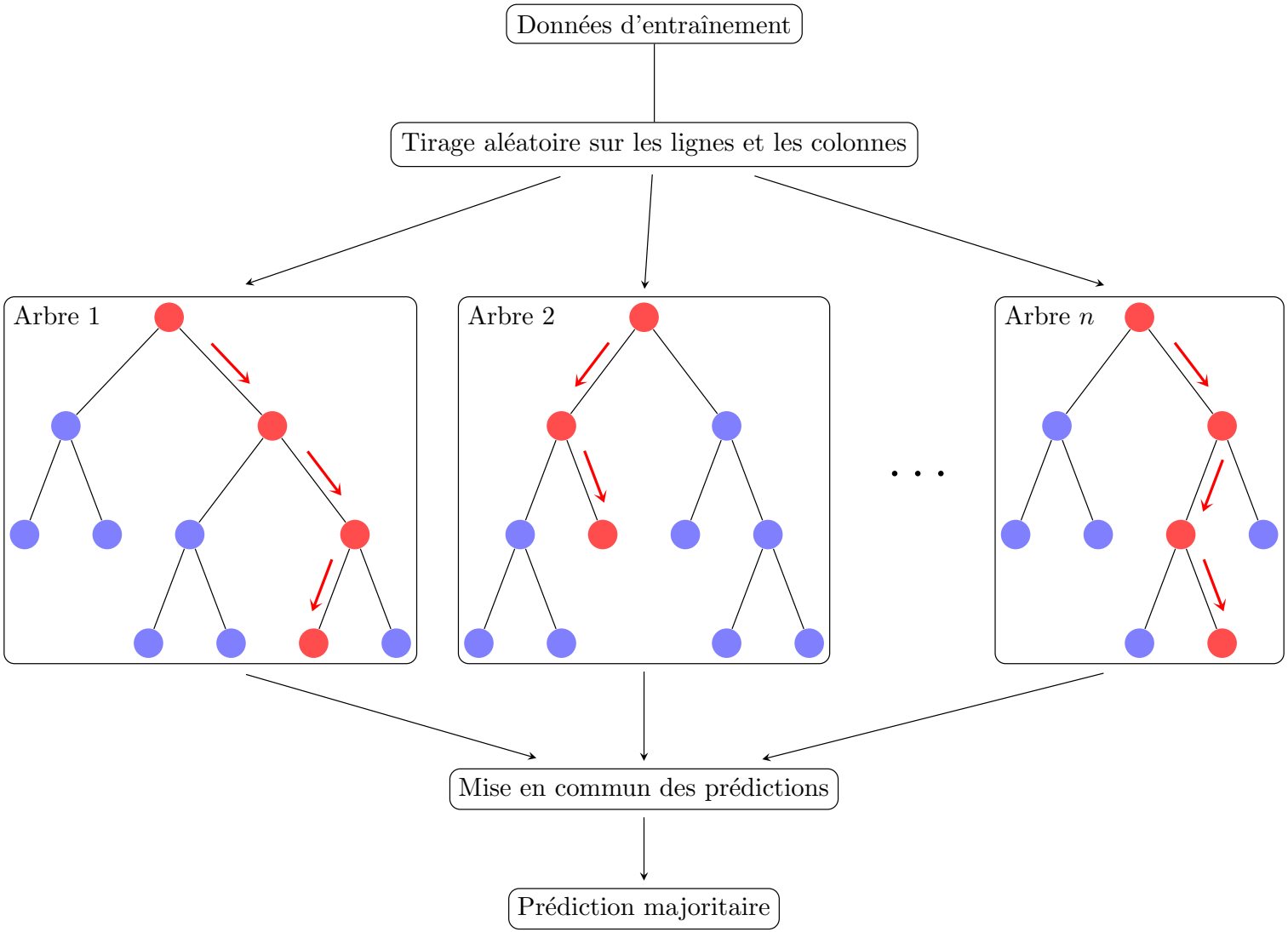

Figure 9: Explication du Random Forest Classifier

Score (précision) du modèle entraîné avec mon classement et ses meilleurs hyperparamètres :

0.68

Score (précision) du modèle entraîné avec le classement ATP et ses meilleurs hyperparamètres :

0.66

Remarque

$$\text{précision} = \frac{\text{nombre de bonnes prédictions}}{\text{nombre total de prédictions}}$$

Evaluation des modèles

Matrices de confusion

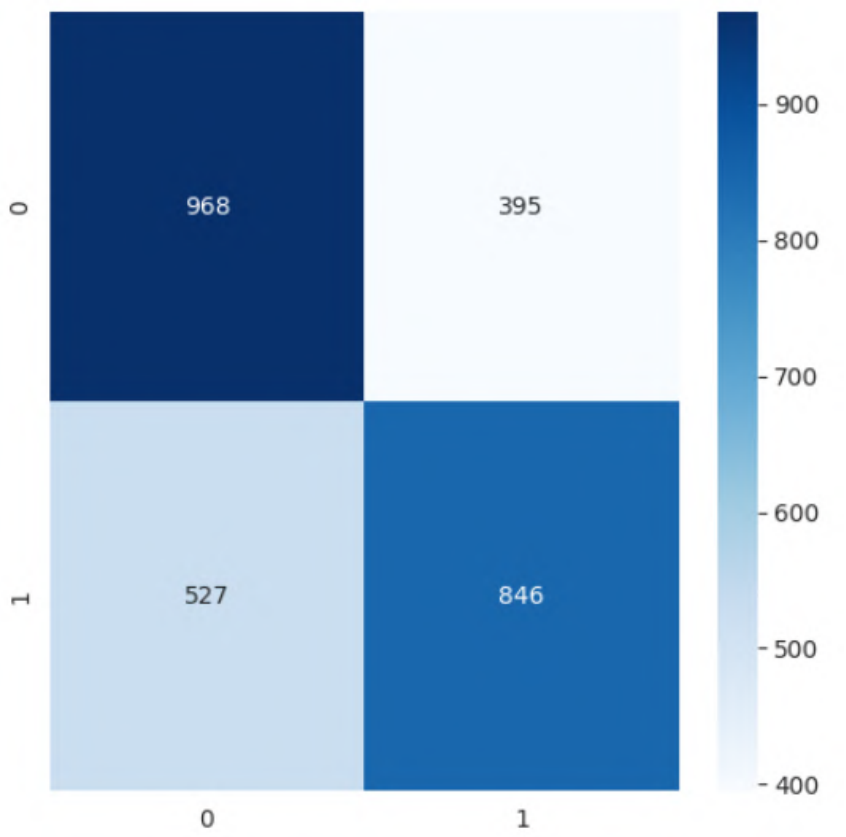

Figure 10: Matrice de confusion classement ATP

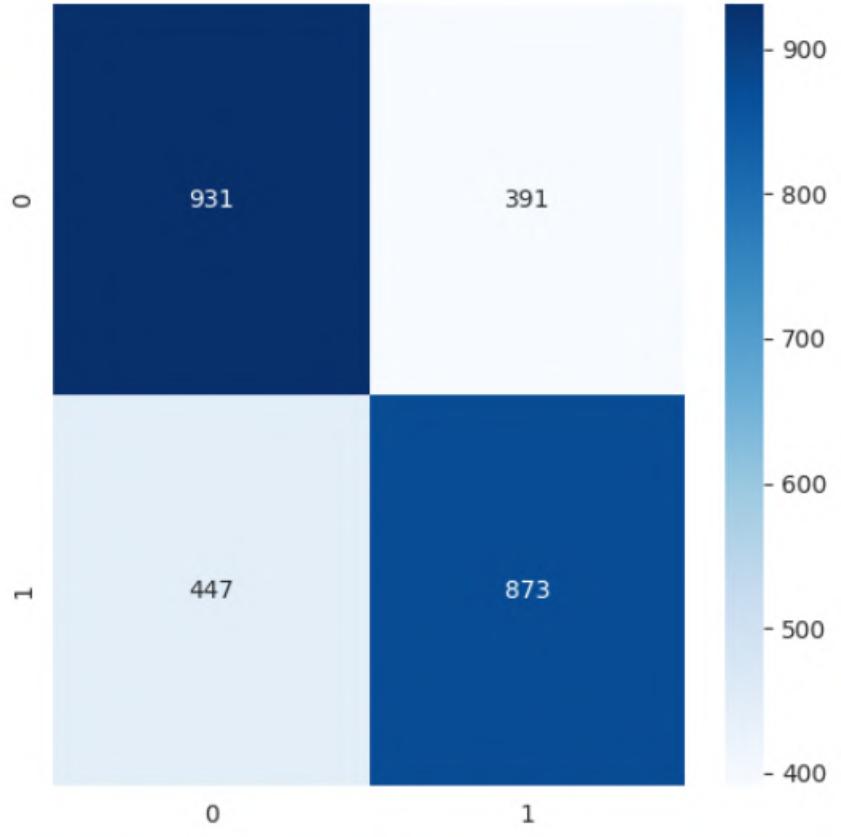

Figure 11: Matrice de confusion mon classement

Evaluation des modèles

"Lucidité" des modèles

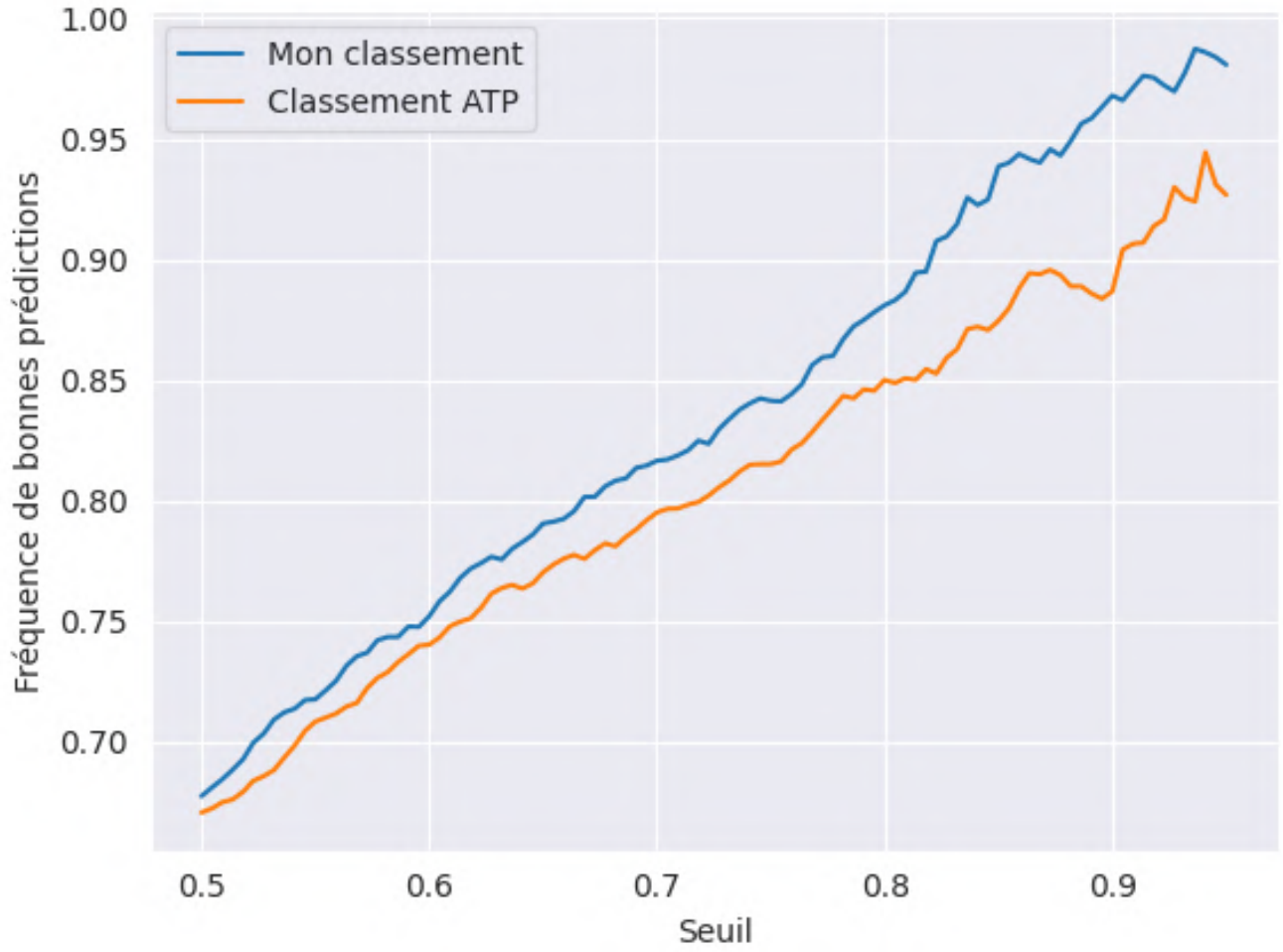

Figure 12: Fréquence de bonnes prédictions en fonction du seuil de probabilité

Le travail de mon binôme

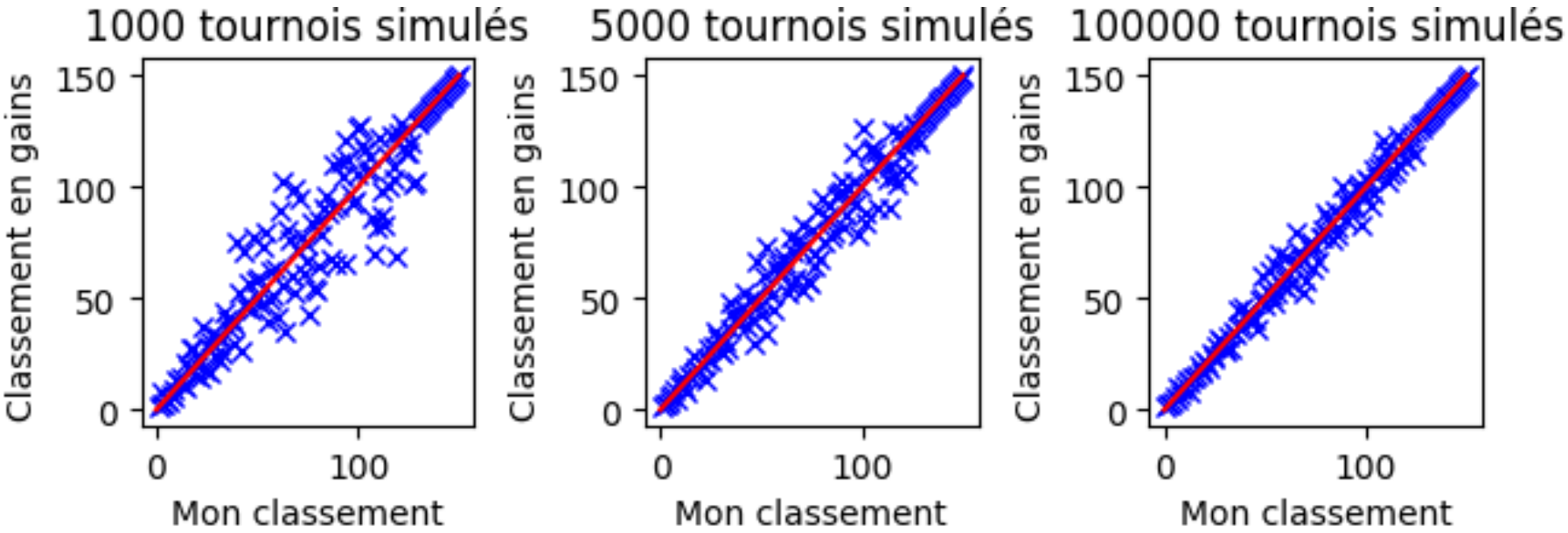

Figure 13: Classement après les tournois en fonction de notre classement

Le travail de mon binôme

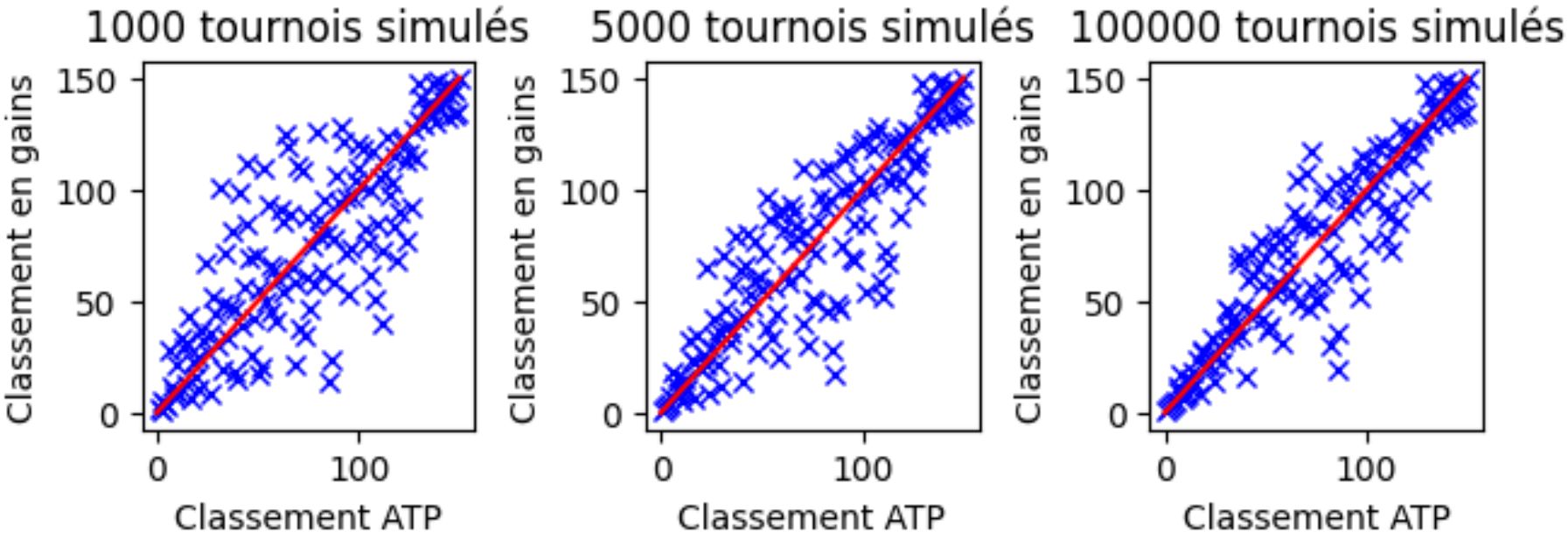

Figure 14: Classement après les tournois en fonction du classement ATP

Conclusion

Soit $P \in \mathcal{GL}_n(\mathbb{C})$ telle que

$$P^{-1}AP = \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & \lambda_1 I_{m_1} + N_1 & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & \lambda_s I_{m_s} + N_s \end{pmatrix}$$

où $\mathrm{Sp}_{\mathbb{C}}(A) = \{1, \lambda_1, \ldots, \lambda_s\}$ et $m_1, \ldots, m_s$ les multiplicités respectives de $\lambda_1, \ldots, \lambda_s$.

$$A^k = P \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & (\lambda_1 I_{m_1} + N_1)^k & & 0 \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & 0 & (\lambda_s I_{m_s} + N_s)^k \end{pmatrix} \cdot P^{-1}$$

A est stochastique donc $\rho(A) = 1$ et pour tout $i \in [\![1, s]\!]$, $|\lambda_i| < 1$.

$$(\lambda_i I_{m_i} + N_i)^k = \sum_{j=0}^{r_i-1} \binom{k}{j} \lambda_i^{k-j} N_i^j \quad \text{où} \quad N_i^{r_i} = 0 \quad \text{et} \quad N_i^{r_i-1} \neq 0$$

$$\text{où} \quad \binom{k}{j} \underset{k\to+\infty}{\sim} \frac{k^j}{j!}$$

$$\text{Il vient,} \quad \sum_{j=0}^{r_i-1} \binom{k}{j} \lambda_i^{k-j} N_i^j \underset{k\to+\infty}{\sim} \frac{k^{r_i-1}\lambda_i^{k-(r_i-1)}}{(r_i-1)!} N_i^{r_i-1}$$

Finalement,

$$A^k \xrightarrow[k\to+\infty]{} P \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \cdot P^{-1}$$

D'où,

$$A^k - P \cdot \begin{pmatrix} 1 & 0 & \cdots & 0 \\ 0 & 0 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \cdot P^{-1} \underset{k\to+\infty}{\sim} C \cdot \lambda_0^k \cdot k^{r_0-1}$$

où $|\lambda_0| = \max_{1\leq i\leq s} |\lambda_i|$.

```
13  import pandas as pd
14  import numpy as np
15  import matplotlib.pyplot as plt
16  from google.colab import drive
17  from sklearn import preprocessing
18  from sklearn.preprocessing import OneHotEncoder
19  import seaborn as sns
20  from datetime import timedelta
21  from datetime import *
22  from random import *
23  import copy
24  from math import *
25  from time import *
26  from datetime import timedelta, datetime
27  color = sns.color_palette()
28  sns.set_style('darkgrid')
29
30  drive.mount('/content/drive')
31
32  # Chargement du dataset/supression des colonnes avec des données manquantes
33
34  data = pd.read_csv('/content/drive/MyDrive/TIPE/DATASET_INITIAL.csv')
35  data = data.dropna()
36  data = data.reset_index(drop=True)
37
38  #Convertion des données en type int
39
40  data['Player1Rank'] = pd.to_numeric(data['Player1Rank'])
41  data['Player2Rank'] = pd.to_numeric(data['Player2Rank'])
42  data['Player1Sets'] = pd.to_numeric(data['Player2Sets'])
43  data['Date'] = pd.to_datetime(data.Date)
44
45  winner = data['Winner'].copy()
```

Figure 15: Programme 1

```python
#Player 1 = mieux classé ATP

def rename(data):
  for k in range(data.shape[0]):
    if data.iloc[k,5]>data.iloc[k,12]:
      data.iloc[k,0],data.iloc[k,1]=data.iloc[k,1],data.iloc[k,0]
      data.iloc[k,4],data.iloc[k,11]=data.iloc[k,11],data.iloc[k,4]
      data.iloc[k,5],data.iloc[k,12]=data.iloc[k,12],data.iloc[k,5]
      data.iloc[k,6],data.iloc[k,13]=data.iloc[k,13],data.iloc[k,6]
      data.iloc[k,7],data.iloc[k,14]=data.iloc[k,14],data.iloc[k,7]
  return data

data = rename(data)

data['Winner']= winner

#Colonne Winner : 1 si P1 gagne 0 sino,
def win(data):
  for k in range(data.shape[0]):
    m = min(data.iloc[k, 5], data.iloc[k, 12])
    # Si P1 gagne...
    if data.iloc[k,6]==data.iloc[k, 15]:
      data.iloc[k,15] = 1
    else:
      data.iloc[k, 15] = 0
  return data

data = win(data)

```

Figure 16: Programme 2

```
"""### **Transformation de certaines variables en variables muettes (dummy variables)**"""

#Transformation des variables round et surface en variables muettes
def round_number(x):
    if x == '1st Round':
        return 1
    elif x == '2nd Round':
        return 2
    elif x == '3rd Round':
        return 3
    elif x == '4th Round':
        return 4
    elif x == 'Quarterfinals':
        return 5
    elif x == 'Semifinals':
        return 6
    elif x == 'The Final':
        return 7

data['Round'] = data['Round'].apply(round_number)

dummy_ranks = pd.get_dummies(data['Round'], prefix='Round')

data = data.join(dummy_ranks.loc[:, 'Round_1':])
data[['Round_1','Round_2', 'Round_3',
      'Round_4', 'Round_5', 'Round_6', 'Round_7']] = data[['Round_1.0','Round_2.0', 'Round_3.0','Round_4.0',
'Round_5.0', 'Round_6.0', 'Round_7.0']].astype('int_')
data = data.drop(['Round_1.0','Round_2.0', 'Round_3.0','Round_4.0', 'Round_5.0', 'Round_6.0', 'Round_7.0'],axis =
1)

dummy_ranks = pd.get_dummies(data['Surface'], prefix='Surface')

data = data.join(dummy_ranks.loc[:, 'Surface_Clay':])
data = data.drop(['Surface'], axis = 1)
data[['Surface_Clay','Surface_Grass','Surface_Hard']] =
data[['Surface_Clay','Surface_Grass','Surface_Hard']].astype('int_')
data = data.drop(["Round"],axis = 1)
```

Figure 17: Programme 3

```
106 """###**Indexation des joueurs : création d'un dictionnaire contenant en clé le nom de joueur et en valeur :
    [index du joueur,son nombre de victoire, total de matchs joués, meilleur classement en carrière]**"""
107
108 ###peut-être une meilleure manière d'encoder les joueurs
109 index_joueurs_stats = {}
110 #Dictionnaire contenant le nom des joueurs comme une clé [player_index,x,y,z,t,w]
111 #x : nombres_de_matchs_gagnés
112 #y : nombres_de_matchs_joués
113 #z : meilleur classement en carrière (on le fixe par défaut à 3000)
114 # t : date du dernier match
115 # w : liste des h2h contre les autres joueurs
116 i=0
117
118 for player in data['Player1'].unique():
119     if player not in index_joueurs_stats.keys():
120         index_joueurs_stats[player] = [i,0,0,3000,0]
121         i+=1
122 for player in data['Player2'].unique():
123     if player not in index_joueurs_stats.keys():
124         index_joueurs_stats[player] = [i,0,0,3000,0]
125         i+=1
126
127 print('Nombre exact de joueurs : ',i)
```

Figure 18: Programme 4

```python
def classement_elo(data,classement,nb_matchs):

  # Création d'une colonne elo_ranking
  elo_p1 = np.zeros((data.shape[0], 1))
  elo_p2 = np.zeros((data.shape[0], 1))
  proba_win_best_ranked = np.zeros((data.shape[0], 1))

  # On parcourt tous les matchs, puis on calcule la proba de gain du joueur le
  # mieux classé contre le moins bien
  for index, row in data.iterrows():
    player1, player2 = row[6], row[11]
    elo1, elo2 = classement[player1], classement[player2]
    K1, K2 = K[player1], K[player2]

    elo_p1[index, 0] = classement[player1]
    elo_p2[index, 0] = classement[player2]


    D = elo1 - elo2
    if abs(D) > 400:
      D = (D/abs(D))*400

    # Proba de gain du joueur le mieux classé sur le pire
    # Proba de gain de P1 sur P2
    p = 1/(1 + 10**(-D/400))

    proba_win_best_ranked[index, 0] = round(p,2)

    if nb_matchs[player1]<=30:
      K[player1] = 40
    elif elo1 >= 2250 or K1==10:
      K[player1] = 10
    else:
      K[player1] = 30

    if nb_matchs[player2]<=30:
      K[player2] = 40
    elif elo2 >= 2250 or K[player2]==10:
      K[player2] = 10
    else:
      K[player2] = 30
```

Figure 19: Programme 5

```
179
180     # Détermination de W, résultat réel
181     W = row[13]
182
183     # Actualisation du classement
184     classement[player1] = round(elo1 + K1*(W-p))
185     classement[player2] = round(elo2 + K2*(p-W))
186
187     nb_matchs[player1] += 1
188     nb_matchs[player2] += 1
189
190
191
192   return (elo_p1, elo_p2, proba_win_best_ranked)
193
194 elo_p1, elo_p2, proba_win_best_ranked = classement_elo(data,classement,nb_matchs)
195
196 # Insertion des colonnes elo_p1 et elo_p2
197 data['elo_P1'] = elo_p1
198 data['elo_P2'] = elo_p2
199 data['proba_elo'] = proba_win_best_ranked
200
```

Figure 20: Programme 6

```
201 """###**Colonne meilleur classement en carrière**"""
202
203 data['Best_rank_p1'] = 0
204 data['Best_rank_p2'] = 0
205
206 for c, row in data.iterrows():
207   p1 = row[6]
208   p2 = row[11]
209   if row[5]<row[27]:
210     data.at[c, 'Best_rank_p1'] = row[5]
211     index_joueurs_stats[p1][3] = row[5]
212   else:
213     data.at[c, 'Best_rank_p1'] = index_joueurs_stats[p1][3]
214
215   if row[10]<index_joueurs_stats[p2][3]:
216     data.at[c, 'Best_rank_p2'] = row[10]
217     index_joueurs_stats[p2][3] = row[10]
218   else:
219     data.at[c, 'Best_rank_p2'] = index_joueurs_stats[p2][3]
```

Figure 21: Programme 7

```
221 """###**Colonnes du pourcentage de victoires (saison en cours)**"""
222
223 index_joueurs_stats_clean = copy.deepcopy(index_joueurs_stats)
224 tableau_ratio = np.zeros(((len(data)),2))
225 date_limite = pd.to_datetime('2007-01-01')
226
227 for c,row in data.iterrows():
228   if row[3]>=date_limite:
229     date_limite = pd.to_datetime(date_limite) + pd.DateOffset(years=1)
230     index_joueurs_stats = copy.deepcopy(index_joueurs_stats_clean)
231   score_p1 = index_joueurs_stats[row[6]]
232   score_p2 = index_joueurs_stats[row[11]]
233   if score_p1[2]<5:
234     tableau_ratio[c,0]=0.2
235   else:
236     tableau_ratio[c,0] =round(score_p1[1]/score_p1[2],2)
237   if score_p2[2]<5:
238     tableau_ratio[c,1]=0.2
239   else:
240     tableau_ratio[c,1] = round(score_p2[1]/score_p2[2],2)
241   if row[13]==1:
242     index_joueurs_stats[row[6]][0],index_joueurs_stats[row[6]][1],index_joueurs_stats[row[6]][2] =
score_p1[0],score_p1[1]+1,score_p1[2]+1
243     index_joueurs_stats[row[11]][0],index_joueurs_stats[row[11]][1],index_joueurs_stats[row[11]][2] =
score_p2[0],score_p2[1],score_p2[2]+1
244   else:
245     index_joueurs_stats[row[6]][0],index_joueurs_stats[row[6]][1],index_joueurs_stats[row[6]][2] =
score_p1[0],score_p1[1],score_p1[2]+1
246     index_joueurs_stats[row[11]][0],index_joueurs_stats[row[11]][1],index_joueurs_stats[row[11]][2] =
score_p2[0],score_p2[1]+1,score_p2[2]+1
247
248 data['ratio_p1_saison'] = tableau_ratio[:,0]
249 data['ratio_p2_saison'] = tableau_ratio[:,1]
```

Figure 22: Programme 8

```
"""## **Colonne du pourcentage de vitoire (en carrière)**"""

index_joueurs_stats = copy.deepcopy(index_joueurs_stats_clean)
tableau_ratio_1 = np.zeros(((len(data)),2))

for c,row in data.iterrows():
  score_p1 = index_joueurs_stats[row[6]]
  score_p2 = index_joueurs_stats[row[11]]
  if score_p1[2]<5:
    tableau_ratio_1[c,0]=0.2
  else:
    tableau_ratio_1[c,0] =round(score_p1[1]/score_p1[2],2)
  if score_p2[2]<5:
    tableau_ratio_1[c,1]=0.2
  else:
    tableau_ratio_1[c,1] = round(score_p2[1]/score_p2[2],2)
  if row[13]==1 :
    index_joueurs_stats[row[6]][0],index_joueurs_stats[row[6]][1],index_joueurs_stats[row[6]][2] =
score_p1[0],score_p1[1]+1,score_p1[2]+1
    index_joueurs_stats[row[11]][0],index_joueurs_stats[row[11]][1],index_joueurs_stats[row[11]][2] =
score_p2[0],score_p2[1],score_p2[2]+1
  else:
    index_joueurs_stats[row[6]][0],index_joueurs_stats[row[6]][1],index_joueurs_stats[row[6]][2] =
score_p1[0],score_p1[1],score_p1[2]+1
    index_joueurs_stats[row[11]][0],index_joueurs_stats[row[11]][1],index_joueurs_stats[row[11]][2] =
score_p2[0],score_p2[1]+1,score_p2[2]+1

data['ratio_p1_carrière'] = tableau_ratio_1[:,0]
data['ratio_p2_carrière'] = tableau_ratio_1[:,1]

```

Figure 23: Programme 9

```python
"""**Fonction qui calcule la proba de gain avec le classement elo**"""

def calcul_proba_elo(pts_elo1, pts_elo2, seuil):
  D = pts_elo1 - pts_elo2
  if abs(D) > seuil:
    D = (D/abs(D))*seuil

  # Proba de gain de P1 sur P2
  p = 1/(1 + 10**(-D/seuil))

  return (p, 1-p)

!pip install fuzzywuzzy

"""**Fonction basé sur la distance de levenstein pour harmoniser les noms des joueurs entre le classement et le dataset**"""

from fuzzywuzzy import fuzz

def abréviation_nom(nom_complet):
    parties_nom = nom_complet.split()
    nom_abrégé = parties_nom[-1]  + ' ' +''.join(parties_nom[0][0])+ '.'
    return nom_abrégé

def trouver_correspondance_nom(nom_inexact, liste_noms):
    correspondance_score_max = 0
    nom_correspondant = None
    for nom_exact in liste_noms:
        score = fuzz.partial_ratio(nom_inexact, nom_exact)
        if score > correspondance_score_max:
            correspondance_score_max = score
            nom_correspondant = nom_exact
    return nom_correspondant

```

Figure 24: Programme 10

```
###cette fonction permet de creer le classement pour une date donnée

def classement_mois(date):
  # Ici on récupère les données relatives à chacun des joueurs à la date directement disponible dans le dataset
  # ie : pour un joueur : [classement_elo, ratio_saison, ratio_carrière]
  stats_joueurs_pour_proba = {}
  date_limite = pd.to_datetime(date)
  date_limite_formatee = date_limite.strftime('%d-%m-%Y')
  data_r = data[(data['Date']>=date_limite-timedelta(days=730)) & (data['Date']<=date_limite_formatee)]
  data_r = data_r.reset_index(drop=True)
  data_r['Date'] = data_r['Date'].dt.strftime('%d-%m-%Y')
  for index, ligne in data_r.iterrows():
    # Données joueur 1
    stats_joueurs_pour_proba[ligne[6]] = [ligne[24], ligne[29], ligne[31]]

    # Données joueur 2
    stats_joueurs_pour_proba[ligne[11]] = [ligne[25], ligne[30], ligne[32]]


  # On importe le classement ATP du 30/10/23 et on récupère les données de chaque joueur (100 premiers à l'ATP)
  # NB : Nécessairement il ne reste donc que les joueurs (ayant joué un match ou plus les deux dernières années)
ET (présent dans le top 100 ATP à la date)
  atp_rkg = pd.read_csv(f'/content/drive/MyDrive/TIPE/classements atp/{date}.csv')
  atp_rkg = atp_rkg[:100]
  stats_joueurs_pour_proba_final = {}
  liste_joueurs = stats_joueurs_pour_proba.keys()
  print(liste_joueurs)
  cpt_index = 0
  for index, row in atp_rkg.iterrows():

    name = row[1]
    l = name.split(' ')
    if len(l) >= 2:
      name = trouver_correspondance_nom(abréviation_nom(name),liste_joueurs)
    if name in liste_joueurs:
      f_stats = stats_joueurs_pour_proba[name]
      # STATS : [index, points ATP, classement ATP actuel, points elo, meilleur classement carrière ATP, Ratio
saison, Ratio carrière]
      stats = [cpt_index, row[4], row[0], f_stats[0], row[5]] + f_stats[1:]
      stats_joueurs_pour_proba_final[name] = stats
      cpt_index += 1
      #print(cpt_index,name,stats,stats_joueurs_pour_proba_final[name])
  print(stats_joueurs_pour_proba_final)
```

Figure 25: Programme 11

```
359   ###Cette fonction ajoute au dictionnaire stats_joueurs_pour_proba_final le nombre de matchs gagnés sur chaque
    autre joueur sous forme d'une liste
360
361     def get_h2h():
362       n = len(stats_joueurs_pour_proba_final)
363
364       for player1 in stats_joueurs_pour_proba_final:
365           p1_h2h = [0 for _ in range(n)]
366           cpt_p2 = 0
367
368           for player2 in stats_joueurs_pour_proba_final:
369             data_pour_h2h = data[((data['Player1'] == player1) & (data['Player2'] == player2 )) | ((data['Player1']
    == player2) & (data['Player2'] == player1))]
370
371             match_p1 = 0
372             for index, row in data_pour_h2h.iterrows():
373               # S'il est player1, mieux classé et winner = 1 OU player2, moins bien classé et winner = 0
374               if (row[6] == player1 and row[13]==1) or (row[11]==player1 and row[13]==0):
375                 match_p1 += 1
376
377             p1_h2h[cpt_p2] = match_p1
378             cpt_p2 += 1
379         stats_joueurs_pour_proba_final[player1].append(p1_h2h)
380
381     get_h2h()
382
```

Figure 26: Programme 12

```python
### Cette fonction calucle la proba de victoire d'un joueur j sur un joueur i en se basant sur la formule du classement élo adaptée à plusieurs statistiques

def calcul_proba(player1, player2):
    stats1, stats2 = stats_joueurs_pour_proba_final[player1], stats_joueurs_pour_proba_final[player2]

    # Calcul proba elo
    p1_elo, p2_elo = calcul_proba_elo(stats1[3], stats2[3], 400)

    # Calcul proba ATP
    p1_atp, p2_atp = calcul_proba_elo(stats1[1], stats2[1], 3000)

    # Calcul proba classement ATP
    p1_catp, p2_catp = calcul_proba_elo(stats1[2], stats2[2], 40)

    # Calcul proba ratio saison
    p1_rs, p2_rs = calcul_proba_elo(stats1[5], stats2[5], 0.4)

    # Calcul proba h2h
    h2h_tot = stats1[7][stats2[0]] + stats2[7][stats1[0]]
    if h2h_tot >= 5:
        p1_h2h, p2_h2h = calcul_proba_elo(stats1[7][stats2[0]], stats2[7][stats1[0]], 10)

        # Calcul proba finale
        p = (50*p1_elo + 25*p1_rs + 25*p1_h2h)/100
    else:
        p = (70*p1_elo + 30*p1_rs)/100

    return (p, 1-p)
```

Figure 27: Programme 13

```python
### Création de la matrice de transition
dict_index_nom = {stats_joueurs_pour_proba_final[player][0]:player for player in stats_joueurs_pour_proba_final}
print(dict_index_nom)

n = len(stats_joueurs_pour_proba_final)
matrice_transition = [[0 for _ in range(n)] for _ in range(n)]
for j in range(n):
    somme = 0

    # On remplit la colonne j
    for i in range(n):
        if i !=j :
            pji = calcul_proba(dict_index_nom[j], dict_index_nom[i])[0]
            matrice_transition[i][j] = pji
            somme += pji
        else :
            matrice_transition[i][j] = 0


    #On normalise la colonne
    for i in range(n):
        matrice_transition[i][j] /= somme

    matrice_transition = np.array(matrice_transition)



n = 100 # dimension de la matrice de transition

```

Figure 28: Programme 14

```python
# Initialisation du vecteur
Y = np.array([[1/n] for _ in range(n)])

# Itération jusqu'à convergence
for i in range(1000):
    tmp = np.dot(matrice_transition, Y)
    Y = tmp / np.linalg.norm(tmp)

# Calcul de la valeur propre dominante
valeur_propre_dominante = np.dot(tmp.T, Y) / np.dot(Y.T, Y)
Y_normalized = Y / np.sum(Y)

# Triez les indices des états en fonction des valeurs du vecteur propre
classement_indices = np.argsort(Y_normalized, axis=0)[::-1]

# Création du classement
classement = [(indice, Y_normalized[indice][0]) for indice in classement_indices]

C = []
for i in range(len(classement)):
  C.append(dict_index_nom[classement[i][0][0]])

return(C)
```

Figure 29: Programme 15

```python
### Cette fonction permet finalement creer le dictionnaire souhaité
def obtenir_classements_personnels():
    classements_personnels = {}
    date_debut = datetime(2015, 1, 1)
    date_fin = datetime(2023, 10, 1)
    mois_actuel = date_debut

    while mois_actuel <= date_fin:
        # Vérifier si le mois actuel est dans la période à sauter
        if mois_actuel.year == 2020 and mois_actuel.month >= 4 and mois_actuel.month <= 8:
            # Passer au mois suivant sans rien faire
            mois_suivant = mois_actuel.replace(month=mois_actuel.month + 1)
            mois_actuel = mois_suivant
            continue

        # Calculer le premier jour du mois suivant
        mois_suivant = mois_actuel.replace(day=1)
        if mois_suivant.month == 12:
            mois_suivant = mois_suivant.replace(year=mois_suivant.year + 1, month=1)
        else:
            mois_suivant = mois_suivant.replace(month=mois_suivant.month + 1)

        # Obtenir le classement des joueurs pour le premier jour de ce mois
        date_str = mois_actuel.strftime('%d-%m-%Y')
        classement = classement_mois(date_str)

        # Ajouter ce classement à la liste des classements mensuels
        classements_personnels[mois_actuel.strftime('%Y-%m')] = classement

        # Passer au mois suivant
        mois_actuel = mois_suivant

    return classements_personnels

# Obtention des classements mensuels
classements_personnels = obtenir_classements_personnels()
```

Figure 30: Programme 16

```python
### je ne garde dans le dataset initial que les matchs à partir du 01-01-2015

data = data[data['Date']>='2015-01-01']
data = data.reset_index(drop=True)

#Supression des matchs impliquant des joueurs au delà de la 100ème place ATP(ceux qui n'ont pas de classement)
for c, row in data.iterrows():
  if (row[5]>100 or row[10]>100):
    data = data.drop([c], axis = 0)
data = data.reset_index(drop=True)

"""# **Création d'un nouveau dataset dans lequel les colonnes classement sont remplies avec mon classement**"""

import pandas as pd
from datetime import datetime

def mettre_a_jour_classements(data, classements_personnels):
    # Créer une copie du dataset original pour le nouveau dataset
    data_my_rank = data.copy()

    # Convertir la colonne de dates en format datetime
    data_my_rank.iloc[:, 3] = pd.to_datetime(data_my_rank.iloc[:, 3])

    # Fonction pour obtenir le classement d'un joueur à une date donnée
    def obtenir_classement_joueur(joueur, date_match):
        date_str = date_match.strftime('%Y-%m')
        classement_mois = classements_personnels.get(date_str, [])
        try:
            classement_joueur = len(classement_mois) - classement_mois.index(joueur)
        except ValueError:
            classement_joueur = float('NaN')
        return classement_joueur
```

Figure 31: Programme 17

```
538
539     # Mettre à jour les colonnes de classement des joueurs dans le nouveau dataset
540     for index, row in data_my_rank.iterrows():
541         date_match = row[3]
542         nom_joueur1 = row[6]
543         nom_joueur2 = row[11]
544         classement_joueur1 = obtenir_classement_joueur(nom_joueur1, date_match)
545         classement_joueur2 = obtenir_classement_joueur(nom_joueur2, date_match)
546         data_my_rank.at[index, 'Player1Rank'] = classement_joueur1
547         data_my_rank.at[index, 'Player2Rank'] = classement_joueur2
548
549     return data_my_rank
550
551 data_my_rank = mettre_a_jour_classements(data, classements_personnels)
552
553 data = data.dropna()
554 data = data.reset_index(drop=True)
555
556 data_my_rank = data_my_rank.dropna()
557 data_my_rank = data_my_rank.reset_index(drop=True)
```

Figure 32: Programme 18

```
# Chargement du data_atpset/supression des colonnes avec des données manquantes

data_atp = pd.read_csv('/content/drive/MyDrive/TIPE/DATASET_ATP_FINAL.csv')
data_atp = data_atp.dropna()
data_atp = data_atp.reset_index(drop=True)

data_myrank = pd.read_csv('/content/drive/MyDrive/TIPE/DATA_MYRANK_FINAL.csv')
data_myrank = data_myrank.dropna()
data_myrank = data_myrank.reset_index(drop=True)

#Convertissement des données en type int

data_atp['Player1Rank'] = pd.to_numeric(data_atp['Player1Rank'])
data_atp['Player2Rank'] = pd.to_numeric(data_atp['Player2Rank'])
data_atp['Player1Sets'] = pd.to_numeric(data_atp['Player2Sets'])
data_atp['Date'] = pd.to_datetime(data_atp.Date)

data_myrank['Player1Rank'] = pd.to_numeric(data_myrank['Player1Rank'])
data_myrank['Player2Rank'] = pd.to_numeric(data_myrank['Player2Rank'])
data_myrank['Player1Sets'] = pd.to_numeric(data_myrank['Player2Sets'])
data_myrank['Date'] = pd.to_datetime(data_myrank.Date)
```

Figure 33: Programme 19

```python
###peut-être une meilleure manière d'encoder les joueurs
index_joueurs_stats = {}
#Dictionnaire contenant le nom des joueurs comme une clé [player_index,x,y,z,t,w]
#x : nombres_de_matchs_gagnés
#y : nombres_de_matchs_joués
#z : meilleur classement en carrière (on le fixe par défaut à 3000)
# t : date du dernier match
# w : liste des h2h contre les autres joueurs
i=0

for player in data_myrank['Player1'].unique():
    if player not in index_joueurs_stats.keys():
        index_joueurs_stats[player] = [i,0,0,3000,0]
        i+=1
for player in data_myrank['Player2'].unique():
    if player not in index_joueurs_stats.keys():
        index_joueurs_stats[player] = [i,0,0,3000,0]
        i+=1

print('Nombre exact de joueurs : ',i)

data_myrank['Best_rank_p1'] = 0
data_myrank['Best_rank_p2'] = 0

for c, row in data_myrank.iterrows():
  p1 = row[6]
  p2 = row[11]
  if row[5]<index_joueurs_stats[p1][3]:
    data_myrank.at[c, 'Best_rank_p1'] = row[5]
    index_joueurs_stats[p1][3] = row[5]
  else:
    data_myrank.at[c, 'Best_rank_p1'] = index_joueurs_stats[p1][3]

  if row[10]<index_joueurs_stats[p2][3]:
    data_myrank.at[c, 'Best_rank_p2'] = row[10]
    index_joueurs_stats[p2][3] = row[10]
  else:
    data_myrank.at[c, 'Best_rank_p2'] = index_joueurs_stats[p2][3]
```

Figure 34: Programme 20

```python
"""**On mélange aléatoirement P1 et P2**"""

import pandas as pd
import numpy as np

def randomize_players_and_winner(df):
    # Sélectionner les colonnes correspondant aux caractéristiques des joueurs
    player_cols = ['Player1', 'B365Player1', 'B365Player2', 'Player1Rank', 'Player2Rank',
                   'Player2', 'elo_P1', 'elo_P2', 'Best_rank_p1', 'Best_rank_p2',
                   'ratio_p1_saison', 'ratio_p2_saison', 'ratio_p1_carrière', 'ratio_p2_carrière']

    # Créer une copie du data_atpFrame pour ne pas modifier l'original
    df_copy = df.copy()

    # Pour chaque ligne du data_atpFrame
    for index, row in df_copy.iterrows():
        # Inverser aléatoirement les colonnes correspondant aux caractéristiques des joueurs
        if np.random.rand() < 0.5:
            # Inverser les colonnes
            df_copy.at[index, 'Winner'] = 1 - df_copy.at[index, 'Winner']
            for col in player_cols:
                col_index = df_copy.columns.get_loc(col)
                opponent_col = col.replace('1', '2').replace('2', '1')
                opponent_col_index = df_copy.columns.get_loc(opponent_col)
                df_copy.iloc[index, col_index], df_copy.iloc[index, opponent_col_index] = df_copy.iloc[index,
opponent_col_index], df_copy.iloc[index, col_index]

    return df_copy

data_atp = randomize_players_and_winner(data_atp)
data_myrank = randomize_players_and_winner(data_myrank)
```

Figure 35: Programme 21

```
128 """**Dernière modifications des datasets...**"""
129
130 # Suppression des colonnes qui ne seront plus utiles au modèle
131 data_atp = data_atp.drop(['Player1','Player2','Player1Sets','Player2Sets','Date'],axis = 1)
132 data_myrank = data_myrank.drop(['Player1','Player2','Player1Sets','Player2Sets','Date'],axis = 1)
133
134 data_atp['Winner'] = pd.to_numeric(data_atp['Winner'])
135 data_myrank['Winner'] = pd.to_numeric(data_myrank['Winner'])
136
137 data_atp = data_atp.drop(['Series'], axis = 1)
138 data_myrank = data_myrank.drop(['Series'], axis = 1)
139
140 data_atp['Player1Pts'] = data_atp['Player1Pts'].astype('int64')
141 data_atp['Player2Pts'] = data_atp['Player2Pts'].astype('int64')
142 data_atp['Player1Rank'] = data_atp['Player1Rank'].astype('int64')
143 data_atp['Player2Rank'] = data_atp['Player2Rank'].astype('int64')
144 data_atp['elo_P1'] = data_atp['elo_P1'].astype('int64')
145 data_atp['elo_P2'] = data_atp['elo_P2'].astype('int64')
146
147
148 data_myrank['Player1Pts'] = data_myrank['Player1Pts'].astype('int64')
149 data_myrank['Player2Pts'] = data_myrank['Player2Pts'].astype('int64')
150 data_myrank['Player1Rank'] = data_myrank['Player1Rank'].astype('int64')
151 data_myrank['Player2Rank'] = data_myrank['Player2Rank'].astype('int64')
152 data_myrank['elo_P1'] = data_myrank['elo_P1'].astype('int64')
153 data_myrank['elo_P2'] = data_myrank['elo_P2'].astype('int64')
```

Figure 36: Programme 22

```
"""###**DIVISION DES DATASETS POUR L'ENTRAINEMENT ET LE TEST**"""

def pre_processing(df):
  df = df.dropna()

  X = df.drop(['Winner'], axis = 1)
  y = df['Winner']

  return X, y

from sklearn.model_selection import train_test_split
trainset_atp, testset_atp = train_test_split(data_atp, test_size = 0.2, random_state = 45)
trainset_myrank, testset_myrank = train_test_split(data_myrank, test_size = 0.2, random_state = 45)

X_train_atp, y_train_atp = pre_processing(trainset_atp)
X_train_myrank, y_train_myrank = pre_processing(trainset_myrank)

X_test_atp, y_test_atp = pre_processing(testset_atp)
X_test_myrank, y_test_myrank = pre_processing(testset_myrank)

print(y_test_myrank.shape)

"""###**MODELISATION**"""

from sklearn import metrics
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import classification_report
from sklearn.model_selection import GroupKFold
from sklearn.model_selection import RandomizedSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.neural_network import MLPRegressor
from sklearn.model_selection import learning_curve
from sklearn.metrics import accuracy_score
```

Figure 37: Programme 23

```
207 from sklearn.tree import DecisionTreeClassifier
208 from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier
209 from sklearn.naive_bayes import GaussianNB
210 from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis,LinearDiscriminantAnalysis
211
212 from sklearn import model_selection
213
214
215 X = X_train_atp
216 Y = y_train_atp
217
218 seed = 42
219
220 models = []
221 models.append(('LR', LogisticRegression()))
222 models.append(('LDA', LinearDiscriminantAnalysis()))
223 models.append(('QDA',QuadraticDiscriminantAnalysis()))
224 models.append(('KNN', KNeighborsClassifier(125, n_jobs=-1)))
225 models.append(('CART', DecisionTreeClassifier(max_depth=10)))
226 models.append(('NB', GaussianNB()))
227
228 models.append(('RandomForest',RandomForestClassifier( n_estimators=100, n_jobs=-1)))
229 models.append(('MLP',MLPClassifier(alpha=0.000001)))
230 models.append(('ADABoost',AdaBoostClassifier()))
231
232
233 results = []
234 scoring = {'accuracy': make_scorer(accuracy_score),
235            'precision_score': make_scorer(precision_score),
236            'recall_score' : make_scorer(recall_score),
237            'f1_score' : make_scorer(f1_score)}
238 names = []
```

Figure 38: Programme 24

```
239 for name, model in models:
240
241     stratifiedKFold = model_selection.StratifiedKFold(n_splits=10, random_state=seed,shuffle = True)
242     cv_results = model_selection.cross_validate(model, X, Y, cv=stratifiedKFold, scoring=scoring)
243     results.append(cv_results)
244     names.append(name)
245     msg
    ='-------------------------------------------------------------------------------------------------------------\n'
246     msg = "Model : %s \n" % (name)
247     msg = msg +'\n'
248
249     msg =  msg + "Accuracy :  %f (%f)\n" % (cv_results['test_accuracy'].mean(), cv_results['test_accuracy'].std())
250     msg =  msg + "Precision score :  %f (%f)\n" % (cv_results['test_precision_score'].mean(),
    cv_results['test_precision_score'].std())
251     msg =  msg + "Recall score :  %f (%f)\n" % (cv_results['test_recall_score'].mean(),
    cv_results['test_recall_score'].std())
252     msg =  msg + "F1 score :  %f (%f)\n" % (cv_results['test_f1_score'].mean(), cv_results['test_f1_score'].std())
253     msg = msg +
    '-------------------------------------------------------------------------------------------------------------\n'
254     print(msg)
255
```

Figure 39: Programme 25

```
"""**GRID SEARCH CV**

**RandomForest**
"""

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

# Définition des hyperparamètres à tester
param_grid = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}

# Création du premier modèle avec le dataset ATP
rf_atp = RandomForestClassifier()

# Création du grid search pour le premier modèle
grid_search_atp = GridSearchCV(estimator=rf_atp, param_grid=param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Entraînement du grid search pour le premier modèle
grid_search_atp.fit(X_train_atp, y_train_atp)

# Meilleurs hyperparamètres pour le premier modèle
best_params_atp = grid_search_atp.best_params_

# Meilleur modèle pour le premier modèle
best_model_atp = grid_search_atp.best_estimator_

# Score du meilleur modèle pour le premier modèle
best_score_atp = grid_search_atp.best_score_

# Affichage des résultats pour le premier modèle
print("Meilleurs hyperparamètres pour le modèle avec classement ATP:", best_params_atp)
print("Score avec les meilleurs hyperparamètres pour le modèle avec classement ATP:", best_score_atp)

# Création du deuxième modèle avec votre propre classement
rf_personnel = RandomForestClassifier()

# Création du grid search pour le deuxième modèle
grid_search_personnel = GridSearchCV(estimator=rf_personnel, param_grid=param_grid, cv=5, scoring='accuracy',
n_jobs=-1)
```

Figure 40: Programme 26

```
300
301 # Entraînement du grid search pour le deuxième modèle
302 grid_search_personnel.fit(X_train_myrank, y_train_myrank)
303
304 # Meilleurs hyperparamètres pour le deuxième modèle
305 best_params_personnel = grid_search_personnel.best_params_
306
307 # Meilleur modèle pour le deuxième modèle
308 best_model_personnel = grid_search_personnel.best_estimator_
309
310 # Score du meilleur modèle pour le deuxième modèle
311 best_score_personnel = grid_search_personnel.best_score_
312
313 # Affichage des résultats pour le deuxième modèle
314 print("\nMeilleurs hyperparamètres pour le modèle avec votre propre classement:", best_params_personnel)
315 print("Score avec les meilleurs hyperparamètres pour le modèle avec votre propre classement:",
    best_score_personnel)
316
317 random_forest_myrank = RandomForestClassifier(max_depth = 10, min_samples_leaf = 3, min_samples_split = 5,
    n_estimators =  25)
318 random_forest_myrank.fit(X_train_myrank,y_train_myrank)
319 print(random_forest_myrank.score(X_test_myrank,y_test_myrank))
320 print(random_forest_myrank.score(X_train_myrank,y_train_myrank))
321
322 # Prédire les probabilités pour le modèle avec votre propre classement
323 probas_myrank = random_forest_myrank.predict_proba(X_test_myrank)[:, 1]
324 predictions_myrank = random_forest_myrank.predict(X_test_myrank)
325
326
327 random_forest_atp = RandomForestClassifier(max_depth = 10, min_samples_leaf = 4, min_samples_split = 10,
    n_estimators =  200)
328 random_forest_atp.fit(X_train_atp,y_train_atp)
329 print(random_forest_atp.score(X_train_atp,y_train_atp))
330 print(random_forest_atp.score(X_test_atp,y_test_atp))
331 # Prédire les probabilités pour le modèle avec classement ATP
332 probas_atp = random_forest_atp.predict_proba(X_test_atp)[:, 1]
333 predictions_atp = random_forest_atp.predict(X_test_atp)
334
```

Figure 41: Programme 27

Annexe

Code

```python
"""Lucidité des modèles"
"""

import numpy as np
import matplotlib.pyplot as plt

# Définir une liste de seuils
seuils = np.linspace(0.5, 0.95, 100)

# Initialiser une liste pour stocker les fréquences de bonnes prédictions
freq_bonnes_predictions_myrank = []

# Pour chaque seuil, calculer la fréquence de bonnes prédictions
for seuil in seuils:
    # Filtrer les prédictions avec des probabilités éloignées de 0,5
    indices = ((probas_myrank < 1-seuil) | (probas_myrank > seuil)).flatten()

    # Sélectionner les prédictions et les vraies étiquettes correspondantes
    score_filtre = accuracy_score(y_test_myrank[indices], predictions_myrank[indices])
    freq_bonnes_predictions_myrank.append(score_filtre)



# Initialiser une liste pour stocker les fréquences de bonnes prédictions
freq_bonnes_predictions_atp = []

# Pour chaque seuil, calculer la fréquence de bonnes prédictions
for seuil in seuils:
    # Filtrer les prédictions avec des probabilités éloignées de 0,5
    indices = ((probas_atp < 1-seuil) | (probas_atp > seuil)).flatten()

    # Sélectionner les prédictions et les vraies étiquettes correspondantes
    score_filtre = accuracy_score(y_test_atp[indices], predictions_atp[indices])
    freq_bonnes_predictions_atp.append(score_filtre)


freq_bonnes_predictions_atp,freq_bonnes_predictions_myrank = np.array(freq_bonnes_predictions_atp),
np.array(freq_bonnes_predictions_myrank)
ecart = np.abs(freq_bonnes_predictions_myrank - freq_bonnes_predictions_atp)
# Calculer l'écart moyen
ecart_moyen = np.mean(ecart)

print(f"L'écart moyen entre les deux courbes est : {ecart_moyen}")

# Tracer la courbe de la fréquence de bonnes prédictions en fonction des seuils
plt.plot(seuils, freq_bonnes_predictions_myrank, label = 'Mon classement')
plt.plot(seuils, freq_bonnes_predictions_atp, label = 'Classement ATP')
plt.xlabel('Seuil')
plt.ylabel('Fréquence de bonnes prédictions')
plt.grid(True)
plt.legend()
plt.show()
```

Figure 42: Programme 28