GEANGOULT Florian

ETOUATI Naïm

Groupe A

Dossier de Modélisation

Sujet : Apprentissage machine et systèmes de recommandation

Table des matières

Introduction :

Nous avons choisi l'apprentissage machine et systèmes de recommandation car c'est un sujet avec lequel nous étions familiers en tant qu'utilisateurs notamment à travers de plateformes comme Netflix. Il était donc particulièrement intéressant mais aussi plus aisé d'appréhender l'objectif du projet.

Le sujet consiste à mettre en place un système de filtrage qui attribue une note « imaginaire » qu'un utilisateur pourrait donner à un item **i** en fonction de la similarité de sa notation d'autres items par rapport à d'autres utilisateurs. Ainsi un utilisateur **a** dont les notes attribuées sont similaires à celles d'un autre utilisateur **b** est susceptible d'attribuer des notes proches de celles de l'utilisateur **b** à un item qu'il n'a pas encore noté.

Pour reprendre l'exemple de Netflix, cela signifie qu'un utilisateur **a** qui aime les mêmes contenus qu'un autre utilisateur **b** est susceptible d'avoir les mêmes goûts et d'aimer un contenu par l'utilisateur **b** qu'il n'a pas encore vue et donc notée. A partir de ces données, il est recommandé à l'utilisateur **a** les contenus pour lesquels sa note « imaginaire » est élevée.

Notre objectif a donc été de calculer la similarité entre les utilisateurs et d'en déduire une note approximative qu'un utilisateur **j** pourrait attribuer à un item **i**. Concrètement, nous avions un jeu de données **complet** représentant les notes, **allant de 0 à 5**, attribuées à **1000** items par **100** utilisateurs. Nous disposions d'un second jeu de données **incomplet** semblable au premier mais dont certaines notes, représentées par des **valeurs -1**, avaient été supprimées. Ce sont ces notes qu'il nous fallait remplacer par une valeur estimée, **aussi proche que possible de celles du jeu de données complet** qui nous servait de référence.

Différentes formules et procédés nous permettent d'arriver à ce résultat, il s'agit donc de comparer les différents résultats au **jeu de données complet** et de déduire quelles méthodes et formules sont les plus efficaces.

1re étape : Traitement de jeux de données sous Python

Notre première problématique a été de **permettre à notre algorithme de lire le jeu de données à compléter**. C'est un support que nous n'avions encore jamais rencontré et nous avons passé un certain temps à appréhender son traitement.

La solution que nous avons trouvée est l'utilisation de la bibliothèque **pandas**. Cette bibliothèque permet d'importer et d'associer un jeu de données à une variable. Une fois importé, le jeu de données nous donnait le rendu graphique suivant :

	Unnamed: 0	Unnamed: 1	...	Unnamed: 999	Unnamed: 1000
0	NaN	-1	...	4	-1
1	NaN	-1	...	4	1
2	NaN	-1	...	5	0
3	NaN	-1	...	-1	0
4	NaN	-1	...	3	1
..	...	...	...	...	...
95	NaN	1	...	4	-1
96	NaN	-1	...	-1	-1
97	NaN	-1	...	2	2
98	NaN	4	...	-1	2
99	NaN	1	...	1	4

La console n'affiche que les valeurs extrêmes dû à leur grand nombre.

On remarque que la première colonne est remplie de valeurs « **NaN** » qui correspondent à des valeurs nulles. Cette colonne n'est pas utile puisque complètement vide, et pourrait fausser le traitement des données, nous l'avons donc supprimée**.**

2ème étape : Calcul de la similarité

Grâce à la documentation fournie nous avons pu définir la formule nécessaire à l'agrégation d'une note :

$$N_{predite}(i,j) = \frac{\sum_{k \in K} poids(i,k) n(k,j)}{\sum_{k \in K} poids(i,k)}.$$

On peut déduire de cette formule 2 éléments :

-poids(i,k) qui correspond à la similarité de notation de l'utilisateur **i** et **k**

-n(k,j) qui correspond à la note attribuée à l'item **j** par l'utilisateur **k**

La **note n(k,j)** peut être déduite du jeu de données importé, mais la **similarité poids(i,k)** doit être calculée par une formule. Il s'agit donc de la première partie que l'algorithme doit traiter. Il existe plusieurs formules de calcul de similarité utilisables. Aiguillés par la documentation fournie ainsi que nos recherches, nous avons implémenté deux formules : la formule de **similarité cosinus**, et la formule de **Pearson.**

Similarité cosinus :

La formule de la similarité cosinus est la suivante :

$$S_C(A,B) := \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}}$$

Ici, **Ai** est la note attribuée par l'utilisateur **A** à l'item **i** et **Bi** est la note attribuée par l'utilisateur **B** à l'item **i**. Ainsi, le numérateur est le produit des notes attribuées par les utilisateurs **A** et **B**. Le dénominateur est le produit des normes des notes attribuées par les utilisateurs **A** et **B**.

Il nous fallait donc parcourir chaque note des utilisateurs **A** et **B** grâce à une boucle. Nous avons stocké les valeurs obtenues grâce à la boucle dans des variables avant d'appliquer la formule de similarité cosinus (cf. Annexe 1).

Le résultat de la similarité cosinus est compris dans l'intervalle **[0 , 1]** et plus la valeur obtenue est proche de **1**, plus les utilisateurs ont des notations similaires. Ainsi, une similarité de **1** équivaut à une notation **exactement similaire** entre l'utilisateur **A** et **B**.

Similarité de Pearson :

La formule de la similarité de Pearson est la suivante :

$$similarity(A,B) = \frac{\operatorname{cov}(A,B)}{\sigma_A \sigma_B} = \frac{\sum_{i=1}^{n}(A_i - \overline{A})(B_i - \overline{B})}{\sqrt{\sum_{i=1}^{n}(A_i - \overline{A})^2 (B_i - \overline{B})^2}}$$

Encore une fois, **Ai** est la note attribuée par l'utilisateur **A** à l'item **i** et **Bi** est la note attribuée par l'utilisateur **B** à l'item **i.** $\overline{A}$ est la moyenne des notes de l'utilisateur **A** et $\overline{B}$ est la moyenne des notes de l'utilisateur **B**. Le dénominateur est alors la covariance entre les utilisateurs **A** et **B** et le dénominateur est le produit de leur écart-type. On remarque que cette formule est la même que la formule du **coefficient de corrélation linéaire**.

Il nous a ainsi fallu calculer au préalable la moyenne des notes des deux utilisateurs avant d'appliquer la formule sur les valeurs récupérées à l'issu du parcours des notes (cf. Annexe 2).

Tout comme pour la similarité cosinus, plus la valeur obtenue par la similarité de Pearson **tend vers 1**, plus les utilisateurs pris en compte ont une notation similaire.

Résultats :

Dans les deux cas, il a fallu faire attention à ne pas prendre en compte les notes non définies **-1** pour ne pas fausser le calcul (cf. ligne 118 Annexe 1 et ligne 118 Annexe 2). Ainsi, si au moins un des deux utilisateurs n'avait pas attribué de note à un item **i**, cet item n'était pas pris en compte lors du calcul de similarité.

Nous avons ensuite stocké les similarités obtenues dans 2 jeux de données pour éviter à l'algorithme de relancer le calcul de similarités à chaque utilisation. Les rendus graphiques obtenus sont les suivants, où **[A,B]** est égal à la similarité entre l'utilisateur **A** et l'utilisateur **B** :

Similarité cosinus entre chaque utilisateur :

	0	1	2	...	97	98	99
0	1.000000	0.729866	0.730757	...	0.752921	0.700979	0.687911
1	0.729866	1.000000	0.731497	...	0.709125	0.966410	0.742882
2	0.730757	0.731497	1.000000	...	0.726275	0.740446	0.740486
3	0.723254	0.742574	0.961687	...	0.758601	0.723983	0.741793
4	0.698780	0.747350	0.954475	...	0.746477	0.754241	0.739672
..	...	...	...	...	...	...	...
95	0.711882	0.742242	0.960864	...	0.754164	0.735852	0.755076
96	0.738404	0.967582	0.732950	...	0.673610	0.959595	0.758859
97	0.752921	0.709125	0.726275	...	1.000000	0.710821	0.702796
98	0.700979	0.966410	0.740446	...	0.710821	1.000000	0.757478
99	0.687911	0.742882	0.740486	...	0.702796	0.757478	1.000000

Similarité de Pearson entre chaque utilisateur :

	0	1	2	...	97	98	99
0	1.000000	-0.021535	0.019100	...	0.073731	-0.079030	-0.138069
1	-0.021535	1.000000	-0.016862	...	-0.094277	0.873937	-0.015483
2	0.019100	-0.016862	1.000000	...	0.028659	-0.018729	0.050737
3	-0.003077	-0.019490	0.857312	...	0.116511	-0.030379	0.019196
4	-0.035760	0.042267	0.830396	...	0.047637	0.045950	0.033650
..	...	...	...	...	...	...	...
95	-0.050733	-0.004248	0.852591	...	0.056090	-0.049384	0.070875
96	0.020283	0.879378	-0.011356	...	-0.116615	0.851127	0.066283
97	0.073731	-0.094277	0.028659	...	1.000000	-0.060138	-0.089225
98	-0.079030	0.873937	-0.018729	...	-0.060138	1.000000	0.042661
99	-0.138069	-0.015483	0.050737	...	-0.089225	0.042661	1.000000

On remarque que pour toute similarité **[A,A]** des deux jeux de données, la valeur obtenue est **1**. En effet, **tout utilisateur A est exactement similaire à lui-même dans sa notation**, et le calcul de sa similarité à lui-même renvoie donc forcément **la valeur 1**. La présence de ces valeurs dans nos résultats permet d'établir que la logique de l'algorithme est correcte.

3ème étape : Calcul de l'agrégation de notes

Rappel de la formule :

$$N_{predite}(i,j) = \frac{\sum_{k \in K} poids(i,k) n(k,j)}{\sum_{k \in K} poids(i,k)}.$$

Une fois le calcul de similarités terminé en utilisant les deux formules précédentes, il est possible d'appliquer la formule d'agrégation de notes. Ainsi chaque absence de note dans le jeu de données (valeurs -1) doit être remplacée par une note estimée en comparant la similarité de l'utilisateur dont la note est manquante à tous les autres utilisateurs.

Pour ce faire, l'algorithme parcourt chaque note du jeu de données, lorsqu'il trouve une valeur **-1**, il la remplace par le résultat de la formule d'agrégation de notes, sinon il passe à la note suivante (cf. Annexe 3). Il est important de noter que les valeurs estimées doivent être implémentées dans un second jeu de données que celui utilisé pour le calcul des notes, afin de ne pas prendre en compte les valeurs précédemment estimées par l'algorithme qui pourraient fausser le calcul. Nous n'avions tout d'abord pas pris ce paramètre en compte ce qui produisait des résultats inexacts.

La formule en elle-même est donc calculée séparément du parcours de valeur, et elle renvoie la note estimée que l'utilisateur **j** pourrait attribuer à l'item **i**. Afin d'arriver à ce résultat, elle récupère les valeurs de similarités stockées précédemment ainsi que les notes non nulles (**-1**) attribuées à l'item **i** par les autres utilisateurs.

La fonction en Annexe 4 prend en compte les valeurs de similarité obtenues avec la **similarité cosinus** mais le principe est exactement le même pour la **similarité de Pearson**.

4ème étape : Correction de la sévérité

Ce filtrage basé sur la similarité entre les utilisateurs présente le défaut de **ne pas prendre en compte la sévérité de l'utilisateur**. En effet, les notes d'un utilisateur qui a tendance à attribuer des notes généralement **basses** ne sont pas forcément de bons paramètres à prendre en compte pour établir une estimation car **sa sévérité fausse le calcul**, alors que ses goûts pourraient être similaires à des utilisateurs dont les notes sont plus élevées, et inversement pour un utilisateur qui aurait tendance à attribuer des notes trop **hautes**.

	Item peu apprécié	Item très apprécié
Utilisateur 1	3	8
Utilisateur 2	1	5
Utilisateur 3	6	10

← Utilisateur sévère

← Utilisateur peu sévère

Dans le tableau ci-dessus, les 3 utilisateurs ont des notations similaires mais une **sévérité différente,** s'il nous fallait estimer une note pour l'utilisateur 1 sur un grand nombre d'items par exemple, la différence de sévérité pourrait affecter le résultat.

Afin de prendre en compte ce paramètre, il est nécessaire de traiter la **note moyenne** des utilisateurs dans le calcul. On obtient ainsi une nouvelle version de la formule de l'agrégation de notes :

$$N'_{predite}(i,j) = \overline{n}(i) + \frac{\sum_{k \in K} poids(i,k)[n(k,j) - \overline{n}(j)]}{\sum_{k \in K} poids(i,k)}.$$

Où $\overline{n}$ est la note moyenne d'un utilisateur.

Un problème algorithmique majeur qu'a posé cette nouvelle formule est **la durée d'exécution de l'algorithme**. Notre première implémentation de la formule calculait les notes moyennes de tous les utilisateurs **à chaque tentative de prédiction de note** pour les prendre en compte dans la correction de sévérité. Il en résultait une **efficacité grandement amoindrie** du programme qui passait alors de nombreuses heures à estimer les notes.
La solution de ce problème a été **un calcul au préalable de la note moyenne de chaque utilisateur**, puis un stockage des valeurs obtenues dans un jeu de données exploitable ultérieurement. Chaque fois que l'algorithme avait besoin d'une note moyenne, il allait **directement la chercher dans ce jeu de données** sans avoir à la calculer à nouveau (Ligne 82 et 85 Annexe 4). Nous avons également implémenté cette méthode **dans l'algorithme de la similarité de Pearson**, qui calculait les notes moyennes jusqu'ici (Ligne 119, 120, et 121 Annexe 2).

On peut alors compléter les notes non attribuées dans notre jeu de données, en se basant sur les deux tableaux de similarité calculés précédemment.

Résultats :

Jeu de données des notes estimées où poids(i,k) = similarité cosinus entre i et k et [A, j] est la note attribuée par l'utilisateur A à l'item j :

```
notes avec cosinus :
    Unnamed: 1  Unnamed: 2  ...  Unnamed: 999  Unnamed: 1000
0          3.0         3.0  ...           4.0            2.0
1          3.0         2.0  ...           4.0            1.0
2          3.0         3.0  ...           5.0            0.0
3          2.0         3.0  ...           3.0            0.0
4          2.0         3.0  ...           3.0            1.0
..         ...         ...  ...           ...            ...
95         1.0         4.0  ...           4.0            2.0
96         3.0         1.0  ...           3.0            2.0
97         3.0         4.0  ...           2.0            2.0
98         4.0         3.0  ...           3.0            2.0
99         1.0         4.0  ...           1.0            4.0
```

Jeu de données des notes estimées où poids(i,k) = similarité de Pearson entre i et k et [A, j] est la note attribuée par l'utilisateur A à l'item j :

```
notes avec pearson :
    Unnamed: 1  Unnamed: 2  ...  Unnamed: 999  Unnamed: 1000
0          2.0         3.0  ...           4.0            4.0
1          4.0         2.0  ...           4.0            1.0
2          1.0         5.0  ...           5.0            0.0
3          1.0         5.0  ...           4.0            0.0
4          0.0         5.0  ...           3.0            1.0
..         ...         ...  ...           ...            ...
95         1.0         4.0  ...           4.0            1.0
96         4.0         1.0  ...           3.0            1.0
97         5.0         5.0  ...           2.0            2.0
98         4.0         1.0  ...           3.0            2.0
99         1.0         4.0  ...           1.0            4.0
```

On peut déjà observer que certaines valeurs diffèrent, mais il est compliqué de comparer à l'œil nu ces résultats à ceux du **jeu de données complété.**

5ème étape : Calcul de la qualité

Maintenant que nous disposons de nos résultats finaux de prédiction de notes, il nous faut les comparer au **jeu de données complet** afin de déduire **quelle formule de similarité a été la plus efficace dans le traitement du sujet**. Pour cela nous disposons de deux formules :

-La formule de calcul du biais :

$$\frac{1}{Q}\sum N_{predite}(i,j) - n_{vraie}(i,j)$$

-La formule de calcul de l'erreur moyenne :

$$\frac{1}{Q}\sum |N_{predite}(i,j) - n_{vraie}(i,j)|$$

Où Q est le nombre de valeurs prédites.

Le biais est **une mesure de la précision des valeurs**, plus il tend vers **0**, plus les valeurs obtenues **correspondent aux valeurs attendues**.

L'erreur moyenne calcule la différence moyenne de la note estimée à la véritable note, c'est un indicateur **de combien de points l'algorithme se trompe généralement lors du calcul des données**. Logiquement, tout comme le biais, **une erreur moyenne de 0 signifie que la prédiction correspond parfaitement aux valeurs attendues**.

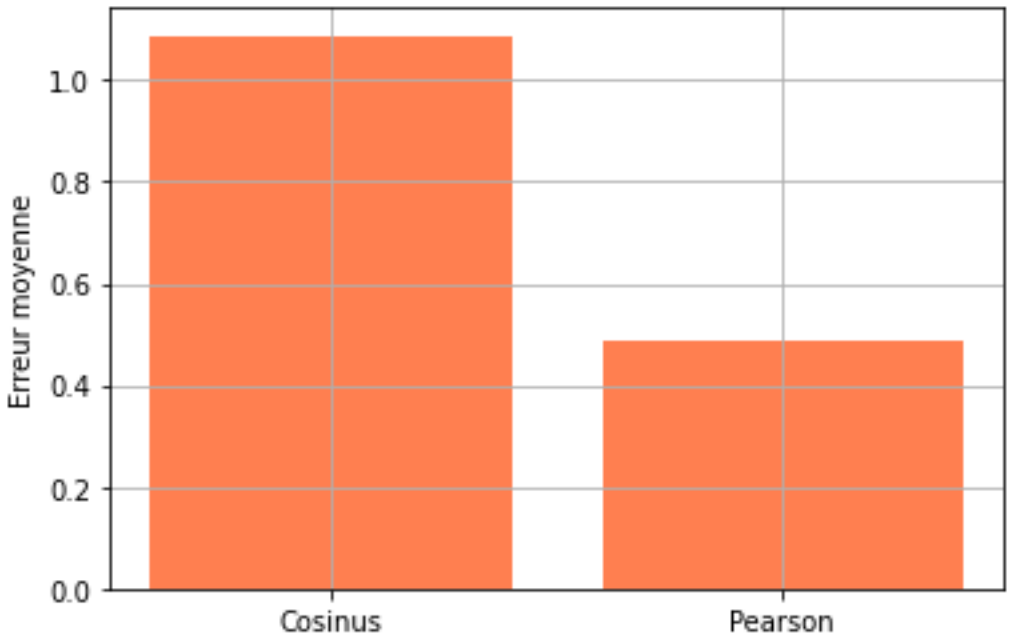
Les deux formules sont similaires et prennent en compte la somme des différence des **notes du jeu de données estimé Npredite(i,j)** aux **notes du jeu de données complet nvraie(i,j).** La différence est que le calcul de l'erreur moyenne traite la valeur absolue de la différence, ce qui permet de connaître la marge d'erreur.

Nous avons donc implémenté le calcul de ces deux valeurs dans deux fonctions différentes par souci de lisibilité, une pour la **similarité cosinus** et une pour la **similarité de Pearson** (Annexe 5 et 6)**.**

Résultats :

```
Biais avec cosinus :  -0.008166103715477846
Erreur moyenne cosinus :  1.0874428770117226
```

```
Biais avec Pearson :  -0.006397774687065368
Erreur moyenne Pearson :  0.48487979336379894
```



Ces résultats nous permettent de facilement déduire **l'efficacité** de nos deux formules et de les **comparer**. La similarité de Pearson présente un biais **moins important** que la similarité cosinus d'environ **0,0022**, et une erreur moyenne environ **2x moins grande** (~1 contre ~0,5).

On peut également questionner l'efficacité de ces deux formules par rapport **à la situation dans laquelle elles sont utilisées**. Nos jeux de données traitent des notes **de 0 à 5**, l'erreur moyenne de la **similarité cosinus** est **d'environ 1**, pour un si petit intervalle, un point représente **1/6 de l'échelle totale**. Cette erreur moyenne reste tolérable mais trop peu efficace pour recommander un item à un utilisateur. Si nous recommandons à un utilisateur **i** tous les items **j** dont sa note estimée serait **>= 4** par exemple, certaines notes pourraient passer de **4 à 3** et l'item concerné ne serait **pas recommandé à l'utilisateur** qui serait pourtant susceptible de l'apprécier, et inversement pour un item que l'utilisateur ne pourrait apprécier que moyennement dont la note passerait de **3 à 4** et **qui lui serait donc recommandé**.
La **similarité de Pearson** cependant présente une erreur moyenne **juste inférieure à 0,5**, ce qui représente **1/12** de l'échelle totale. En sachant que la notation ne prend en compte que **des entiers**, cette erreur moyenne peut être acceptable grâce à **l'arrondi des notes**. **C'est donc une formule bien plus intéressante que la similarité cosinus à prendre en compte dans notre cas.**

Conclusion

	Biais	Erreur moyenne
Similarité cosinus	-0.008166	1.087
Similarité de Pearson	-0.006398	0.485

Nos résultats finaux nous montrent que la similarité de Pearson est une formule bien plus intéressante à utiliser dans notre cas que la similarité cosinus. Son biais et son erreur moyenne sont bien plus proches de 0, et elle est d'autant plus efficace si on l'évalue par rapport à l'utilisation que nous en faisons, où son erreur moyenne est négligeable.

Cette étude des systèmes de recommandation nous a permis de comprendre l'importance de la comparaison entre différentes méthodes de calcul. Bien que différentes méthodes soient utilisables, leur efficacité diffère entre elles mais aussi par rapport à la situation dans laquelle elles sont utilisées. Il en résulte que l'emploi de certaines méthodes et formules peut nuire à la précision d'une déduction statistique.

De plus, et malgré que ce ne soit pas un des domaines les plus approfondis par notre sujet, la comparaison entre différentes implémentations algorithmiques est également primordiale. Nous nous sommes retrouvés plusieurs fois face à des algorithmes dont le temps d'exécution était trop long pour s'avérer efficace, il nous a alors fallu revoir la logique du programme tout en obtenant les mêmes résultats. L'exemple le plus flagrant de cela aura été l'implémentation du calcul des notes moyennes, qui dans un cas prenait des heures voir des jours à achever son algorithme contre quelques minutes à peine une fois reprogrammé.

Enfin, ce projet nous aura permis de nous projeter dans quelque chose de concret, et montré à quoi peut servir notre code dans des conditions plus ou moins réelles, ce qui est très enrichissant pour nous qui sommes habitués au théorique. Nous avons ainsi pu faire le lien entre mathématiques et programmation, nous donnant du recul sur le domaine de l'informatique en général.

Annexe 1 : Fonction de calcul de similarité cosinus

```
112     def cosinus(o,g):
113         dénom1 = 0
114         dénom2 = 0
115         num = 0
116 
117         for j in range(df.shape[1]) :
118                 if(df.iloc[o,j] != -1 and df.iloc[g,j] != -1) :
119                         dénom1 += (df.iloc[o,j])**2
120                         dénom2 += (df.iloc[g,j])**2
121                         num +=(df.iloc[o,j]*df.iloc[g,j])
122 
123         dénom2 = sqrt(dénom2)
124         dénom1 = sqrt(dénom1)
125         sci = num/(dénom1*dénom2)
126         sci = round(sci,6)
127 
128         return sci
```

Annexe 2 : Fonction de calcul de similarité de Pearson

```
112     def pearson(o,g) :
113 
114         num =0
115         dénom1=0
116         dénom2=0
117         for j in range (df.shape[1]):
118             if(df.iloc[o,j]!=-1 and df.iloc[g,j]!=-1):
119                 num += (df.iloc[o,j]-moyennetab.iloc[o,0])*(df.iloc[g,j]-moyennetab.iloc[g,0])
120                 dénom1 += (df.iloc[o,j]-moyennetab.iloc[o,0])**2
121                 dénom2 += (df.iloc[g,j]-moyennetab.iloc[g,0])**2
122 
123 
124         denom = sqrt(dénom1*dénom2)
125         simPearson=num/denom
126         return simPearson
```

Annexe 3 : Fonction de recherche de valeurs -1 dans le jeu de données

```
39      def parcoursValeurs():
40 
41          for i in range(df.shape[0]):
42 
43              for j in range(df.shape[1]):
44 
45 
46                  if df.iloc[i,j] == -1:
47                      print("l'utilisateur " , i , " a n'a pas de note colonne : " , j )
48                      print(df.iloc[i,j])
49                      IncompletComplété.iloc[i,j] = predite(i,j)
50                      print(IncompletComplété.iloc[i,j])
```

Annexe 4 : Fonction de prédiction d'une note de l'item j pour l'utilisateur k

```
66  def predite(k, j):
67
68
69      simTotal = 0
70      prediteNum = 0
71
72
73      for i in range(df.shape[0]):
74
75
76
77          if df.iloc[i,j] != -1:
78
79              simTotal += simtabcos.iloc[k,i]
80              #simTotal += simtabpearson.iloc[k,i]
81
82              prediteNum += (simtabcos.iloc[k,i]*(df.iloc[i,j]-moyennetab.iloc[i,0]))
83              #prediteNum += (simtabpearson.iloc[k,i]*(df.iloc[i,j] -moyennetab.iloc[i,0]))
84
85      return round(moyennetab.iloc[k,0]+(prediteNum/simTotal),0)
```

Annexe 5 : Calcul de la qualité du tableau déduit par similarité cosinus

```
127  def comparaisonCos():
128
129      somme = 0
130      q = 0
131      sommenorm = 0
132
133      for i in range(df.shape[0]):
134          for j in range(df.shape[1]):
135              if(df.iloc[i,j] == -1):
136                  somme += (completCos.iloc[i,j] - df2.iloc[i,j])
137                  sommenorm += abs(completCos.iloc[i,j] - df2.iloc[i,j])
138
139                  q += 1
140
141      print("q = ", q)
142      print("Biais avec cosinus : ", somme/q)
143      print("Erreur moyenne cosinus : ", sommenorm/q)
144
145      return 0
```

Annexe 6 : Calcul de la qualité du tableau déduit par similarité de Pearson

```
147  def comparaisonPear():
148
149      somme = 0
150      sommenorm = 0
151      q = 0
152
153      for i in range(df.shape[0]):
154          for j in range(df.shape[1]):
155              if(df.iloc[i,j] == -1):
156
157                  somme += (completPear.iloc[i,j] - df2.iloc[i,j])
158                  sommenorm += abs(completPear.iloc[i,j] - df2.iloc[i,j])
159                  q += 1
160
161      print("q = ", q)
162      print("Biais avec Pearson : ",somme/q)
163      print("Erreur moyenne Pearson : ",sommenorm/q)
164      return 0
```

Bibliographie :

https://www.datacamp.com/community/tutorials/recommender-systems-python

https://towardsdatascience.com/intro-to-recommender-system-collaborative-filtering-64a238194a26

https://fr.acervolima.com/python-mise-en-oeuvre-du-systeme-de-recommandation-de-films/#:~:text=Recommender%20System%20est%20un%20syst%C3%A8me,des%20choix%20de%20l%27utilisateur.&text=Ce%20mod%C3%A8le%20est%20ensuite%20utilis%C3%A9,d%27int%C3%A9resser%20l%27utilisateur

https://docs.python.org/3.8/

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.iloc.html

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html