Science des données

IFT799 – TP4

Amazon Book Reviews

UNIVERSITÉ DE SHERBROOKE

Elèves : Tahar AMAIRI et Corentin POMMELEC

Session : Automne 2022

Introduction :

Le code de ce TP est contenu dans un Jupyter Notebook. Pour son exécution, il nécessite plusieurs modules déjà très connus dans le domaine de la science des données : pandas, numpy, matplotlib, seaborn.

Le jeu de données que nous utilisons est celui de l'Amazon Book Reviews (ABR) contenant les revues effectuées par les clients sur les livres qu'ils ont achetés sur Amazon. Celui-ci se trouve dans un fichier JSON contenant différentes informations sur les avis des clients. On y trouve notamment l'ASIN (Amazon Standard Identification Number) du livre, sa note ou encore le reviewerID permettant d'identifier l'utilisateur postant sa revue. Pour charger le fichier JSON contenant toutes les reviews, nous allons le découper en "chunk" afin de pouvoir le lire car il est extrêmement lourd (~10 GB). Pour cela, il est possible en fonction de la puissance de votre ordinateur d'augmenter la taille des chunks. Dans le cadre de la réalisation de ce TP, nous avons chargé tous les avis.

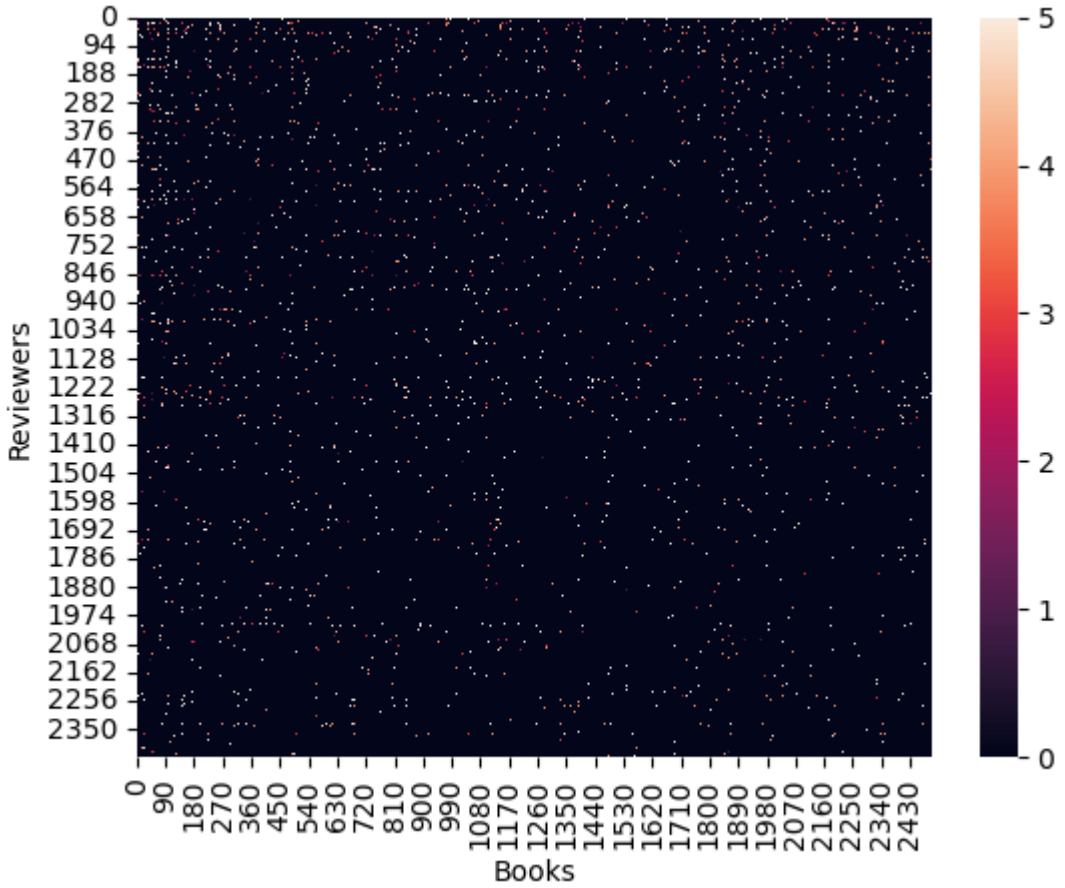
Pour rappel, nous avons commencé dans le TP1 par une exploration des données, qui nous a permis d'analyser le jeu de données et, ainsi, de mieux le comprendre.

Ensuite, dans le TP2&3 on a exploité des caractéristiques statistiques, caractérisant nos données, afin de réaliser une segmentation de ces dernières à partir de plusieurs stratégies de clustering. Le tout avec différentes utilisations des données, stratifiées ou non, desquelles nous avons comparé les résultats obtenus.

Enfin, dans ce dernier travail, le TP4, nous allons effectuer une prédiction basée sur le principe du filtrage collaboratif par évaluation hors-ligne.

a. Créer un dataframe contenant les avis de n utilisateurs (lignes) sur p livres (colonnes), en s'assurant que le nombre de cases vides ne soit pas significatif.
Pour cela, nous commençons par utiliser toutes les données, récupérées du jeu de données ABR, dans la fonction `getFr1`. Avec en paramètre p pour le nombre de livres et n pour le nombre de reviewers, cette fonction commence par trier les livres et les reviewers avec le moins de valeur nulles et, à l'aide de ses paramètres, récupère le nombre d'éléments voulus. Elle crée ensuite le dataframe souhaité avec les p livres, en colonnes, et les n reviewers, en lignes, avec comme valeur leurs notes associées. Nous avons chargé un maximum de livres selon les capacités de notre machine (en terme de RAM), i.e, n = p = 2500.

Ensuite, on affiche une heatmap, permettant d'observer le sous-ensemble de données que l'on a sélectionné :



Heatmap plot Fr1 (avec `plotHeatMap`)

On remarque qu'il y a encore beaucoup de noir dans la heatmap, c'est-à-dire beaucoup de valeur nulle. Cela se remarque également lorsqu'on affiche le pourcentage de valeurs nulles pour chaque livre dans l'ordre décroissant :

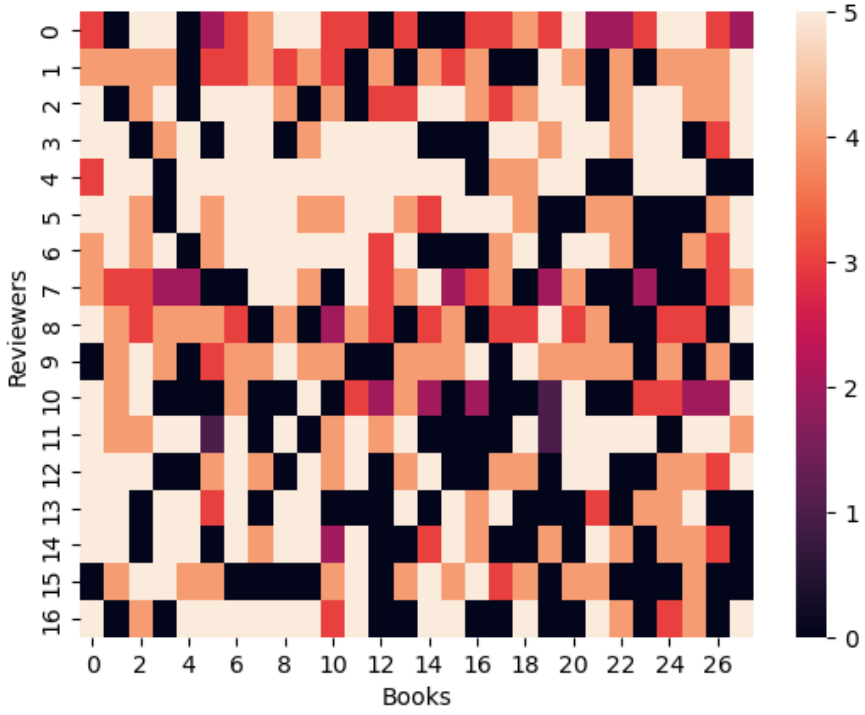
	Percentage of non zero value
0439023483	12.162162
0143170090	9.049959
0439023513	7.944308

Affichage pourcentage de valeurs nulles par livre (avec `plotStats`)

Pour remédier à ce problème, on va essayer de trouver une sous-matrice dense beaucoup plus petite : pour cela, on utilise la fonction `getDenseSubMatrix`. Cette fonction construit une sous matrice dense itérativement en triant la matrice Fr1 selon les meilleurs livres / utilisateurs (en termes d'avis) puis enlève les r livres / utilisateurs les plus mauvais. Elle s'arrête dès que la limite imposée sur la taille est atteinte. Finalement, elle enlève aussi les livres / utilisateurs avec un écart type faible (utile pour les prochaines questions) selon un certain seuil. Dans notre cas, nous avons choisi comme paramètre :

- r = 1
- std = 0.5
- minN = 30
- minP = 30

En choisissant un faible r, on est sûr d'obtenir une sous matrice dense d'une taille au plus 30*30 avec un écart-type d'au moins 0.5. Et cela se confirme avec les différents plots :



Heatmap plot nouveau Fr1 (avec `plotHeatMap`)

	Percentage of non zero value
1476755590	88.235294
0615748996	88.235294
0062294776	82.352941
0989450201	82.352941
1476776016	82.352941
0451468279	76.470588

Affichage pourcentage de valeurs nulles par livre (avec `plotStats`)

b. Créer 3 sous-ensembles d'entraînement à partir de Fr1 pour lesquels chacune des lignes contiennent des ratings cachés aléatoirement.

A l'aide de la fonction `dataSelection` qui prend en paramètre le nombre de ratings à cacher par ligne. On commence par parcourir toutes les lignes et on récupère les index des ratings qui ne sont pas nuls, afin de ne pas masquer des notes déjà nulles. Ensuite pour créer les 3 sous ensembles on se base sur une copie de Fr1 sur laquelle nous cachons aléatoirement le nombre de ratings désirés, aux index récupérés précédemment, si la ligne contient plus de valeurs non nulles que de notes à cacher.

On peut aussi remarquer que le nombre de valeurs non-nulles a augmenté pour chaque dataset obtenu (ce qui est tout à fait logique) :

Dataset 1

	Percentage of non zero value
0615748996	82.352941
0989450201	82.352941
0062294776	76.470588
1476776016	76.470588
1476718202	76.470588
1476755590	70.588235
1492121169	70.588235

Dataset 2

	Percentage of non zero value
0062294776	82.352941
0615748996	82.352941
0989450201	82.352941
1483966763	76.470588
1455548987	76.470588
1492121169	76.470588
006230240X	76.470588

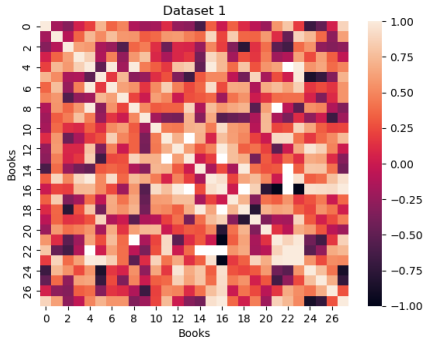
Dataset 3

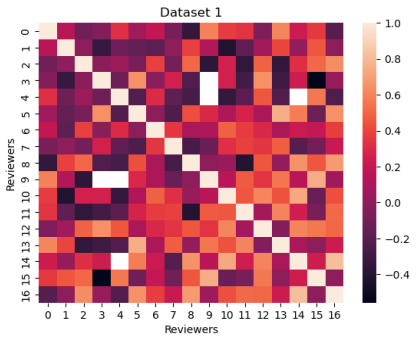
	Percentage of non zero value
1476755590	88.235294
0615748996	88.235294
1476776016	82.352941
0989450201	76.470588
0062294776	76.470588
0988695103	76.470588
061579615X	70.588235

Pour information, nous avons caché 2 avis par ligne afin d'éviter d'avoir des matrices creuses.

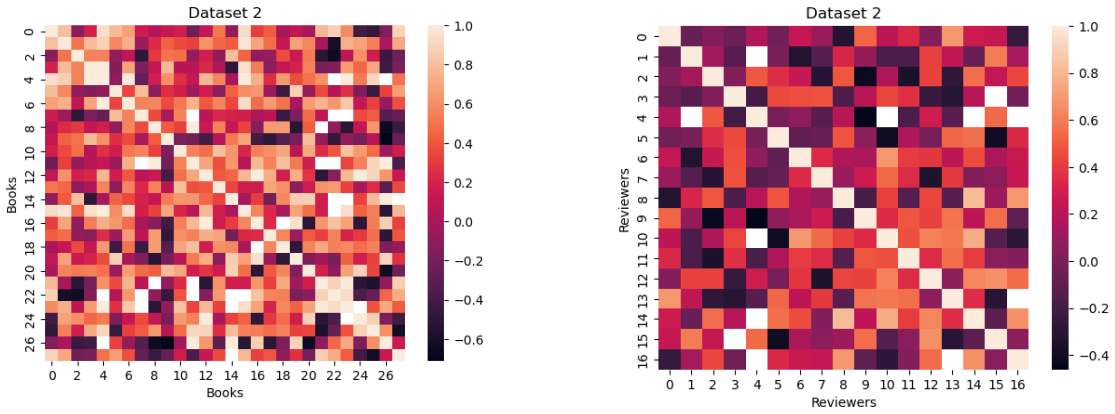
c.1) Calculer les matrices de similitudes des livres et les matrices de similitudes des reviewers de chacun des sous-ensembles créés précédemment.

La fonction `corr` de la librairie pandas permet le calcul de matrices de similitude. Il suffit de lui spécifier la méthode pearson en paramètre pour qu'elle se base sur le coefficient de Pearson lors du calcul des similitudes. En bouclant sur tous les sous-ensembles d'entraînement, la fonction `corr` se base sur les colonnes c'est-à-dire initialement sur les livres pour créer des similitudes entre eux. On obtient ainsi nos premières matrices de similitudes entre les livres. Ensuite, pour obtenir nos matrices de similitudes entre les reviewers, il suffit de fournir à la fonction pandas la transposée de nos dataframes d'entraînement. En effet, cela permet aux reviewers de se situer aux niveaux des colonnes et donc de pouvoir être traités par la fonction de corrélation de pandas. Nous pouvons désormais apprécier les similitudes obtenus à l'aide de l'affichage d'heatmaps pour chaque matrice de similitudes créées.

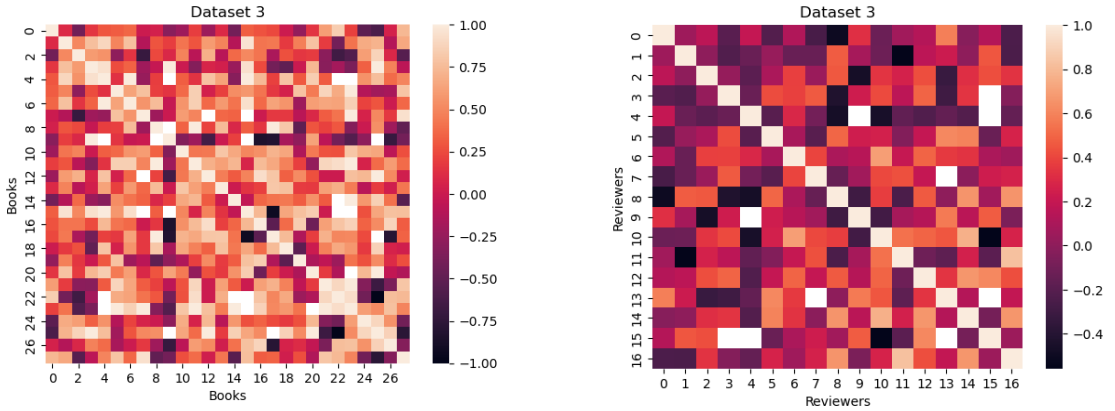




Heatmap livres et utilisateurs dataset 1



Heatmap livres et utilisateurs dataset 2



Heatmap livres et utilisateurs dataset 3

On remarque à l'aide de ces affichages d'heatmap qu'il y a une grande hétérogénéité des similitudes entre les éléments aussi bien pour les livres que pour les reviewers. Cela est cohérent et permettrait de repérer des groupes d'éléments similaires. Globalement,comme voulu ces affichages permettent de rapidement évaluer les similitudes entre deux éléments ou plus.

c.2) A partir des matrices de similitudes créées, pour chaque livre extraire l'ensemble des 3 livres les plus similaires.

Nous utilisons la fonction `extractMostSimilarElements` prenant en paramètre la matrice de similitude de laquelle extraire, l'ensemble des éléments les plus similaires à l'élément concerné, ainsi qu'un autre paramètre contenant la valeur du nombre d'éléments similaires contenus par les ensembles à créer. Cette fonction parcourt tous les éléments de chaque colonne de la matrice et trie par ordre décroissant les valeurs de similitudes. Elle récupère ensuite, les n éléments avec les plus grandes valeurs de similitudes pour chacune des colonnes. Donc dans notre cas nous lui fournissons la matrice de similitudes des livres et lui indiquons de créer pour chacun des livres un ensemble composé des 3 livres les plus similaires à celui concerné, en ne prenant pas en compte la similitude entre le livre et lui-même qui est toujours de 1. L'ensemble formé des 3 plus grandes similitudes est stocké dans un dictionnaire avec en clef l'identifiant du livre. Finalement, nous vérifions aussi si

nous n'avons pas la présence de valeurs NaN à l'aide la fonction `checkNaNPresence` retournant True si la sélection effectuée pour chaque livre contient une valeur NaN.
Ceci sera important pour ne pas fausser l'apprentissage des paramètres lors de la question d). En complément, la vérification se fait à l'aide d'un assert : `assert(checkNaNPresence(listI) == False)`

c.3) A partir des matrices de similitudes créées, pour chaque reviewer extraire l'ensemble des 8 reviewers les plus similaires.
De même, afin d'extraire pour chaque reviewer l'ensemble des 8 reviewers les plus similaires au concerné, nous réutilisons la fonction `extractMostSimilarElements`. Sur le même principe, nous fournissons la matrice de similitudes des reviewers et nous lui indiquons de créer pour chacun des livres un ensemble composé des 8 reviewers les plus similaires à celui concerné, en ne prenant pas en compte la similitude entre le reviewer et lui-même qui est toujours de 1. L'ensemble formé des 8 plus grandes similitudes est stocké dans un dictionnaire avec en clef l'identifiant du reviewer. De même, nous vérifions à l'aide d'un assert si nous n'avons pas sélectionné des valeurs NaN.

d.1) Calculer le gradient de Eval(Ω)
Voir le fichier PDF avec le rapport pour la démonstration.

d.2) Suivant la méthode de descente des gradients, estimer les paramètres Ω.
Pour estimer les paramètres de la prédiction nous allons utiliser trois fonctions :
- `computeAllZSum` : en prenant en paramètre un dataset de test Fr et les ensembles I et U associés à celui-ci (obtenu à la question c), elle permet de calculer une matrice AllZ stockant pour chaque livre i et pour chaque utilisateur j sa somme de matrice Z comme spécifié dans l'énoncé à la fonction R().
- `computeLossGradient` : en prenant en paramètre le vecteur de paramètres W, un ensemble de données de test Fr, cette matrice calcule le gradient de Eval() au point W. Ici, nous avons effectué une modification par rapport à l'énoncé : en effet, on compte le nombre de rij qui ne sont pas nuls (i.e non NaN) pour effectuer la division de la somme par 1 / (n*p) dans Eval(). Par conséquent, la loss est uniquement calculée pour les valeurs rij connues dans Fr.
- `gradientDescent` : en prenant en paramètre un dataset de test Fr, les ensembles I et U associés à celui-ci, un nombre maximum d'itérations maxIter et un taux d'apprentissage eta, elle permet d'effectuer une descente de gradient pour obtenir les paramètres W.

Dans notre cas, nous avons utilisé ces paramètres :
- maxIter = 5000
- eta = 0.001

Nous avons pris ces paramètres pour être certain d'obtenir une convergence (un eta assez faible et beaucoup d'itérations).

d.3) A partir de la valeur de Ω, prédire les sentiments des utilisateurs dont vous avez cachés les valeurs lors des 10 dernières périodes en utilisant la fonction R()

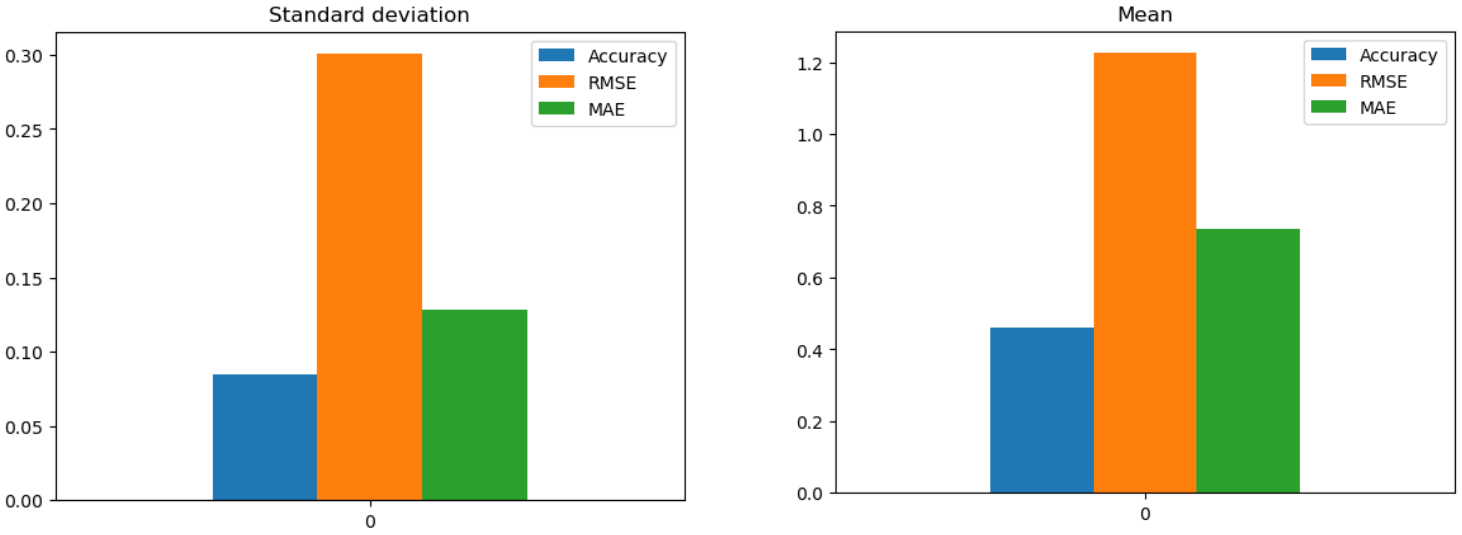
Pour prédire les valeurs cachées, nous allons utiliser la fonction `predict` prenant en paramètres le dataset test Fr à prédire, les ensembles I et U associés à celui-ci, le vecteur W des paramètres obtenu suite à l'entraînement et les ratings cachées à prédire.
Nous avons aussi remarqué que parfois la prédiction retourne des valeurs plus grandes que 5, dans ce cas là, nous limitons la valeur obtenue à 5.0. Finalement, on prend la valeur entière de chaque prédiction pour pouvoir comparer plus simplement avec le système de ratings qu'on a.

d.4) Calculer l'erreur quadratique RMSE et l'erreur absolue moyen MAE

Pour calculer la RMSE et la MAE, nous allons utiliser les fonctions de sklearn directement à l'aide de la fonction `getErrors.` Pour des raisons de compréhensions supplémentaires, nous avons aussi ajouté la précision. Voici, ce que nous obtenons :

	Accuracy	RMSE	MAE
Test1	0.558824	0.882353	0.588235
Test2	0.411765	1.352941	0.823529
Test3	0.411765	1.441176	0.794118

Tableau d'erreur



Barplot de la moyenne et de l'écart type

Nous obtenons une précision moyenne de 47 % ce qui est plutôt décevant car près d'une prédiction sur deux est fausse. Par conséquent, cela se remarque aussi sur la RMSE et la MAE : on se retrouve avec une RMSE assez élevée et une MAE plutôt satisfaisante car en moyenne nous sommes à 75 % d'une rating de différence (car MAE moyenne de 0.73). Concernant l'écart type entre les différentes mesures, il n'est pas assez significatif et cela se voit (nous avons le test2 et test3 qui ont des résultats très similaires).

Finalement, malgré plusieurs améliorations implémentées (utilisation d'une sous-matrice dense, pas de NaN dans le calcul des ensembles I et U, cacher très peu de scores, filtrer les

données de Fr1 avec un critère d'écart-type), le jeu de données reste toujours très mal adapté à la question car on est forcé d'utiliser une petite matrice.
En effet, nous obtenons des résultats très faibles, signe d'un sous-apprentissage de notre modèle, pouvant être expliqué par plusieurs raisons :

- on se base sur une matrice d'une taille 16*27, nous avons donc très peu de données sur lesquelles on peut entraîner notre modèle.
- la descente de gradient n'a pas convergé à cause d'un manque d'itération et/ou d'un mauvais taux d'apprentissage.
- la plupart des livres ont un très bon rating, donc le modèle a tendance à sur-apprendre et à mal prédire les bas ratings.
- ou bien, une erreur dans l'implémentation éventuellement.