Rapport de Pldac - Système de recommendation

ELGUINDY Abdelrahman 21102968
ELHUSSIENY Habiba 21105223
LU Yuchen 21400505

I. Présentation du dataset

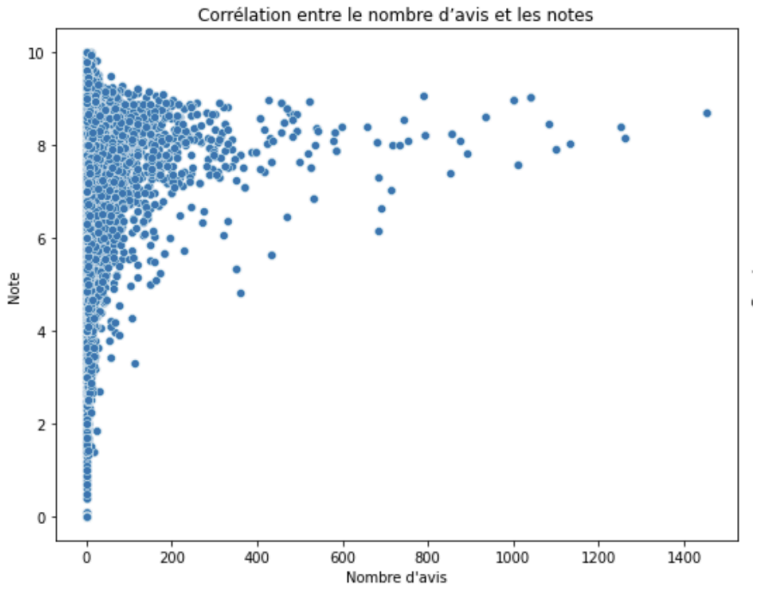
Le dataset contient 246524 avis (notes) dont 81.57% sont accompagnés de commentaires. Il contient les infos sur 20235 jeux dont 67.30% ont au moins un avis.

Ces données essentielles ont été scrappées depuis le site TricTrac et ont été stockées dans quatre fichiers BSON. Celui des avis est le plus utile, avec des notes et des commentaires des utilisateurs, qui constituent la base principale pour la recherche de cette Partie 1. Celui des détails est aussi utile puisqu'il contient des infos catégorielles sur les jeux et leurs diverses notes.

Le dataset a été bien nettoyé. Il y a deux avis spam qui gênent légèrement la qualité de la visualisation, mais n'ont pas d'impact sur les résultats d'analyse.

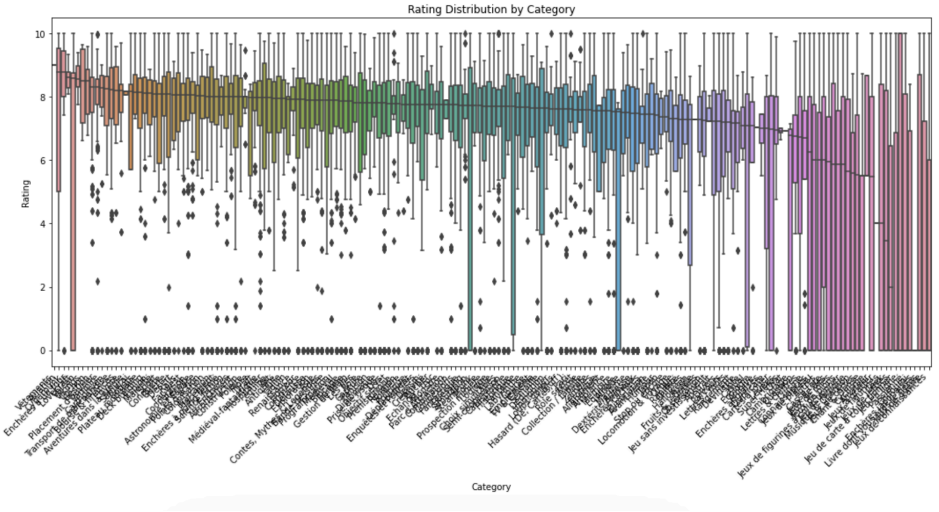
II. EDA : Exploration des données

Parmi les essais d'EDA, deux méritent d'être présentés. Le premier illustre la corrélation entre le nombre d'avis et la note moyenne pour un jeu. Il est clair que plus le nombre d'avis est élevé, plus le jeu a de chances d'obtenir une bonne note moyenne (entre 6 et 9). Ce résultat s'explique intuitivement par le fait que les utilisateurs sont généralement plus motivés à donner leur avis sur les bons jeux.



Le deuxième est un boxplot des notes regroupées par catégories de jeux. Le fait que les valeurs médianes et les quartiles soient globalement similaires suggère que le paramètre « catégorie » ne serait pas très utile dans une approche Content-Based Filtering.

Yuchen : Mon doute persiste que la similarité des médianes ne signifie pas qu'individuellement, les utilisateurs n'ont pas de fortes préférences envers certaines catégories. On pourrait sans doute calculer les variances pour approfondir.



III. Construction de la matrice de notation

Les prochaines étapes de notre recherche consistent à préparer et expérimenter divers modèles pour prédire les notes des utilisateurs. Les données des commentaires sont laissées pour la Partie 2. L'hypothèse ici est que le Collaborative Filtering est plus performant que le Content-Based Filtering. C'est généralement le cas dans le domaine des systèmes de recommandation. Et de plus, ici, les données catégorielles ne semblent pas suffisantes.

Par conséquent, nous créons une ratings_matrix composée de tous les utilisateurs et de tous les jeux. C'est une matrice de 12035 lignes (utilisateurs) et 16366 colonnes (jeux). Le fill-in rate est de 0.08 %, ce qui indique une matrice extrêmement clairsemée. Il faut gérer ce problème avant d'implémenter les modèles.

Au début, on tente une imputation par plusieurs méthodes : la moyenne des notes par jeu, celle par utilisateur, ou encore la distribution normale par utilisateur prenant en compte des infos catégorielles. On réalise ensuite que cette imputation n'a pas de sens. Vu la forte sparsité de la ratings_matrix, cela ne ferait que remplir la matrice avec des données artificielles, rendant tout modèle ML inefficace. Cet essai n'est pourtant pas inutile. Le calcul des moyennes par utilisateur et par jeu, même inadapté pour l'imputation, servira de benchmark pour évaluer les modèles.

Pour augmenter le fill-in rate, on applique une autre méthode : réduire la taille de la matrice en supprimant les utilisateurs ayant peu d'avis et les jeux peu notés. La difficulté ici est le déclenchement de l'effet ping-pong, c'est-à-dire que la suppression de jeux peu notés entraîne la suppression d'utilisateurs (qui perdent leurs rares jeux restants), ce qui entraîne ensuite la suppression d'autres jeux, et ainsi de suite, dans une boucle sans fin.

Nous appliquons une fonction « minipingpong » et retrouvons par plusieurs tests empiriques les seuils minimaux optimaux du nombre de notes par utilisateur (=14) et par jeu (=18). Nous avons également réfléchi à des compromis entre trois critères liés à la réduction de taille (même si ce n'est pas encore un grand souci) : en augmentant le fill-in rate, on risque de détériorer la représentativité des données (i.e. divergence de Kullback-Leibler) et de réduire le taux de rétention.

Yuchen : Je ne suis pas certain que les seuils optimaux ont été testés automatiquement. Si ce n'est pas le cas, on pourrait appliquer la méthode GridSearchCV de la bibliothèque Surprise (ref : RS1) pour automatiser la recherche.

IV. Implémentation des modèles

L'étape suivante est d'implémenter des modèles de benchmark pour préparer l'évaluation. On se base sur RS1 et installe quatre modèles de base : Global Mean, User Mean, Game Mean et BaselineOnly. Ce dernier prend en compte les biais utilisateurs et jeux via leurs moyennes. Comme indiqué dans RS1, ce modèle simple est très puissant et pourrait surpasser des modèles plus avancés. C'est exactement ce que nous constatons dans notre recherche.

V. Comparaison des performances

L'étape la plus importante est de comparer les modèles implémentés. On commence par examiner les résultats de classement. Le second classement intègre plusieurs modèles aux paramètres optimisés.

Model Rankings :

	Model	RMSE Rank	MAE Rank	MSE Rank	R2 Rank	Avg Rank
3	BaselineOnly (Surprise)	1.0	1.0	1.0	1.0	1.00
4	SVD (Surprise)	2.0	2.0	2.0	2.0	2.00
7	KNNWithMeans (Suprise)	3.0	3.0	3.0	3.0	3.00
8	Sklearn KNN	4.0	4.0	4.0	4.0	4.00
6	KNNBasic (Suprise)	5.0	5.0	5.0	5.0	5.00
0	Global Mean	6.0	7.0	6.0	9.0	7.00
1	User Mean	7.0	6.0	7.0	8.0	7.00
2	Game Mean	8.0	8.0	8.0	7.0	7.75
5	NMF (Suprise)	9.0	9.0	9.0	6.0	8.25

```
Final Model Rankings (including optimized models):
                        Model  RMSE Rank  MAE Rank  MSE Rank  R2 Rank  Avg Rank
9   BaselineOnly (Optimized)        1.0       1.0       1.0      1.0      1.00
3    BaselineOnly (Surprise)        2.0       2.0       2.0      2.0      2.00
10  KNNWithMeans (Optimized)        3.0       3.0       3.0      3.0      3.00
4             SVD (Surprise)        4.0       4.0       4.0      4.0      4.00
7     KNNWithMeans (Suprise)        5.0       5.0       5.0      5.0      5.00
11           NMF (Optimized)        6.0       6.0       6.0      6.0      6.00
8               Sklearn KNN        7.0       7.0       7.0      7.0      7.00
6         KNNBasic (Suprise)        8.0       8.0       8.0      8.0      8.00
0                Global Mean        9.0      10.0       9.0     12.0     10.00
1                  User Mean       10.0       9.0      10.0     11.0     10.00
2                  Game Mean       11.0      11.0      11.0     10.0     10.75
5              NMF (Suprise)       12.0      12.0      12.0      9.0     11.25
```

Quatre observations immédiates :

1. Il n'y a pas de différences significatives entre les résultats des différents indicateurs.
2. BaselineOnly est le plus performant, avec une marge notable.
3. La performance de NMF s'améliore considérablement après optimisation.
4. KNNWithMeans est meilleur que KNNBasic.

Ce qui importe le plus est d'analyser en détail l'optimisation des paramètres, l'explicabilité et les limites des modèles.

D'abord, les modèles de benchmark :

1. La performance de BaselineOnly montre que les biais dans les échelles de notation sont un défi majeur dans le domaine de la recommandation.
2. Comme pour les films dans RS1, User Mean surpasse Game (Item) Mean. Cela confirme que les comportements utilisateurs sont plus prédictifs que les tendances des items en situation de démarrage à froid ou de données clairsemées.
3. Le fait que User Mean dépasse Game Mean suggère aussi que le Content-Based Filtering ne fonctionnera pas bien ici, car il s'appuie sur les caractéristiques des jeux, et non des utilisateurs.

Pour les autres modèles, on peut les classer en deux groupes :
- Le premier comprend trois modèles KNN ;
- Le second, SVD et NMF, basés sur la factorisation matricielle.

La comparaison s'effectue principalement au sein de chaque groupe.

Pour les KNN :

1. KNN de Sklearn est le seul modèle parmi les cinq qui ne fait pas partie de la bibliothèque Surprise. Surprise est conçue pour la recommandation, tandis que Sklearn est une bibliothèque généraliste. Elle gère donc moins bien les biais, la logique de prédiction des notes, et elle est plus lourde à programmer.
2. Surprise offre une meilleure explicabilité grâce à sa structure de sortie en namedTuple, plus traçable, et à sa capacité à identifier les utilisateurs ou jeux similaires ayant influencé la prédiction. Il nous reste à explorer davantage cet aspect explicatif.

3. KNNWithMeans, après optimisation, est le deuxième meilleur modèle (après BaselineOnly). Son avantage sur KNNBasic illustre encore une fois l'importance de corriger les biais.
   Yuchen : Il est certain que l'amélioration est liée à l'augmentation de k (nombre de voisins) de 40 à 60. Mais la version optimisée utilise aussi cosine au lieu de pearson pour sim_options. Ce dernier est souvent considéré comme supérieur pour le Collaborative Filtering. Il faudrait tester les performances avec pearson.

Pour les modèles à factorisation matricielle :

1. NMF, avant optimisation, se révèle moins performant que tous les quatre modèles de benchmark. Une hypothèse à vérifier est que NMF partage un problème observé avec Sklearn KNN lors de nos premiers tests : le modèle prédit très mal les notes extrêmement basses. Il peut par exemple prédire un 8 pour une note réelle de 2. Ces erreurs sont lourdement pénalisées par MSE, ce qui semble être le cas pour la première version de NMF (index 5 dans le tableau ci-dessous).

```
Comparison of all models (original vs. optimized):
                        Model      RMSE       MAE       MSE      R2 score
0                 Global Mean  2.082863  1.690738  4.338317 -5.499470e+30
1                   User Mean  2.111329  1.658971  4.457709 -5.686745e+00
2                   Game Mean  2.261259  1.793801  5.113291 -4.707552e+00
3    BaselineOnly (Surprise)  1.774035  1.379631  3.147199  2.739725e-01
4              SVD (Surprise)  1.828462  1.406829  3.343273  2.287403e-01
5               NMF (Suprise)  2.402300  2.040289  5.771045 -3.313226e-01
6         KNNBasic (Suprise)  1.952939  1.537309  3.813972  1.201547e-01
7     KNNWithMeans (Suprise)  1.834945  1.406834  3.367022  2.232615e-01
8                 Sklearn KNN  1.940607  1.507077  3.765956  1.317725e-01
9   BaselineOnly (Optimized)  1.766282  1.362118  3.119753  2.803041e-01
10  KNNWithMeans (Optimized)  1.793888  1.379995  3.218034  2.576316e-01
11           NMF (Optimized)  1.835224  1.436870  3.368046  2.230253e-01
```

2. Après optimisation, la performance de NMF s'améliore nettement, grâce à l'ajustement de n_factors – de 15 à 30 – doublant ainsi le nombre de facteurs latents, ce qui permet de capter des préférences plus complexes entre utilisateurs et jeux. De plus, des régularisations plus fortes (reg_pu et reg_qi) ont été appliquées pour limiter le surapprentissage.
   Yuchen : Changer le paramètre "bias=False" à "bias=True" pourrait encore améliorer la performance de NMF.
3. La contre-performance de NMF avant optimisation vient probablement du fait que, contrairement à SVD, il ne traite que des valeurs positives. Les facteurs ne peuvent donc pas s'annuler entre eux (comme c'est le cas avec les valeurs négatives en SVD). Cela rend les facteurs latents plus traçables et rend le modèle plus explicable.
4. La version optimisée de NMF reste légèrement inférieure à SVD, peut-être parce que NMF est moins expressif pour les avis négatifs. En SVD, ils peuvent être modélisés avec des valeurs négatives. En NMF, on ne peut que réduire la valeur jusqu'à 0, ce qui empêche de différencier un avis négatif d'un avis indifférent. Cela pourrait aussi expliquer pourquoi la première version de NMF a du mal à prédire les notes très basses.